# Bachelor's thesis

# TuberXpert
## The expert system that adjusts treatments

A project of



| | |
|---|---|
| **Student:** | **Melvyn Herzig** |
| **Work proposed by:** | Professor Yann Thoma |
| | REDS |
| | Route de Chesaux 1 |
| | 1401 Yverdon-les-Bains |
| **Responsible teacher:** | **Professor Yann Thoma** |
| **Expert:** | **Jonas Chapuis** |
| **Academic year:** | 2021-2022 |

Yverdon-les-Bains on Friday, July 29, 2022

# Table of contents

Sometimes this document refers to certain directories or files by providing their path. All these indications are given assuming that the reader has access to the root of the repository of this project. For the final submission of this thesis, it has been provided with the document as a compressed file. Otherwise, access to the repository can be requested from *Professor Yann Thoma*.

**Department:** TIC
**Faculty:** Computer science
**Specialization:** Software
**Student:** Melvyn Herzig
**Responsible teacher:** Prof. Yann Thoma
**Expert:** Jonas Chapuis

Bachelor's thesis 2021-2022

TuberXpert

**Publishable summary**

Tanzania is a country with a high incidence of tuberculosis, a deadly airborne lung disease. Every year, 120'000 people contract the disease. However, only half of them receive treatment and most of the time it fails. In one to two-thirds of tuberculosis patients, the peak concentration of the antitubercular drug is below the reference range. In addition, one-third of people with tuberculosis must take antiretroviral drugs that further lower the concentration of the antitubercular drug. The risk of treatment failure is real.

Taking a not suitable treatment is dangerous. The patient may become drug resistant, intoxicated or even die. In consequences, the Tanzanian government wants to end tuberculosis by 2035.

Fortunately, the fight is not lost. The main ally is therapeutic drug monitoring, which is a specialized practice in measuring drug levels in the blood for a treatment. However, the whole process is slow. In addition, it takes an experienced pharmacologist to interpret the samples and recommend an effective adjustment. To facilitate this part, Professor Yann Thoma in partnership with the Lausanne University Hospital, has developed a software called Tucuxi. However, it still needs to be handled by an experienced practitioner.

In view of the current situation, TuberXpert is a project between Switzerland and Tanzania that aims to create a clinical decision support system to democratize access to therapeutic drug monitoring. Based on the Tucuxi computation core, this Bachelor's thesis consists in developing a first version of the software.

Indeed, by obtaining data on the patient, TuberXpert is able to evaluate the plausibility of the received data and to propose a first adapted treatment or an adjustment by providing a complete report in English or French.

Since the software was developed with a basic knowledge of therapeutic drug monitoring, the results are very encouraging. In its final form, with more knowledge of pharmacology, TuberXpert should be a real success.

# 1 <u>Preamble</u>

This Bachelor's thesis is produced at the end of the course of study, with a view to obtaining the title of Bachelor of Science HES-SO in Engineering.

As an academic work, its content, without prejudging its value, does not engage the responsibility of the author, nor that of the jury of the Bachelor's thesis and of the school.

Any use, even partial, of this Bachelor's thesis must be made in compliance with the copyright.

HEIG-VD

The head of the department

Yverdon-les-Bains on Friday, March 4, 2022

# 2 <u>Introduction</u>

## 2.1 What is therapeutic drug monitoring

Nowadays, many drugs or antibiotics are used to treat diseases such as tuberculosis (TB) and human immunodeficiency viruses (HIV). Usually, the doctors prescribe generic doses that are suitable for the general population. Unfortunately, everyone's metabolism reacts differently which makes generic dosages often ineffective.

Some people will have insufficient circulating drug exposure caused by an underdose. Thus, the treatment will be ineffective, and the patient may become drug resistant. Conversely, an overdose may result in intoxication. This would force an interruption of the treatment in order not to worsen the patient's health.

To avoid such situations, therapeutic drug monitoring (TDM) has been developed. This is a precision medicine that prescribes a personalized dosage to each patient based on the monitoring of the evolution and the prediction of the drug concentration in the blood.
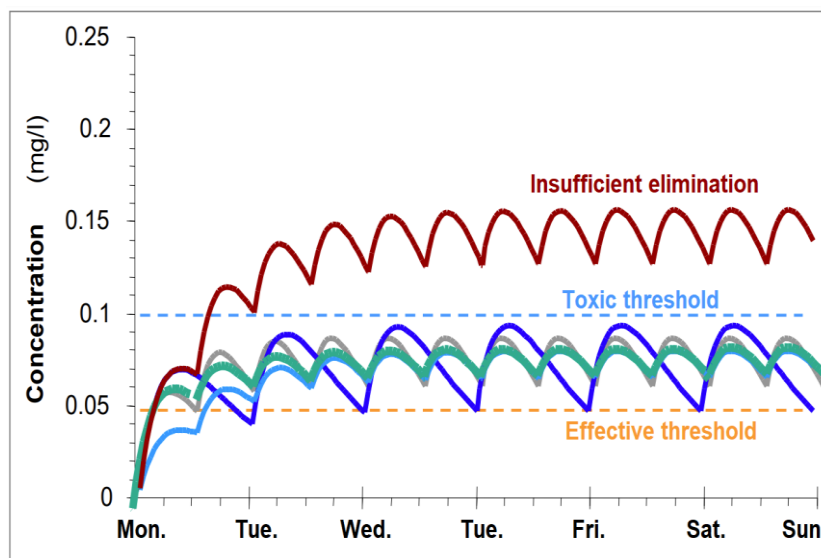


*Figure 1 - Dosage scheme (BUCLIN Thierry, 2022, Les Bases de la pharmacocinétique clinique [PDF document])*

After the administration of a drug, the prediction of the evolution of blood concentrations depends on:

- Methods of administration such as formulation, route of administration and dosages.
- Pharmacokinetic parameters that describe the behavior of the drug in the body. These parameters are influenced by the physical characteristics of the patient, called covariates.

The pharmacokinetic parameters are said to be:

- <u>Typical patient</u>: If generic covariates are used to calculate the parameters.
- <u>A priori</u>: If they exist, the patient's covariates override the default values to calculate the parameters.
- <u>A posteriori</u>: This is like "A priori", but patient samples are also considered for the calculation.

## 2.2 Tucuxi

Currently, Professor Yann Thoma and the Lausanne University Hospital developed <u>Tucuxi</u>. It is a software intended for the practice of TDM. Already developed for several years, the software offers many features:

- Drug concentration predictions based on population and patient data (covariates) as well as on previous measurements (samples).
- Suggestion of dosage adjustments to reach an optimal drug concentration state.

- Generation of reports.
- Integration with electronics health record systems.

Tucuxi is available in 3 formats:

- [A web application](#).
- A graphical user interface (GUI) application for desktop.
- A command-line interface (CLI) application for desktop.
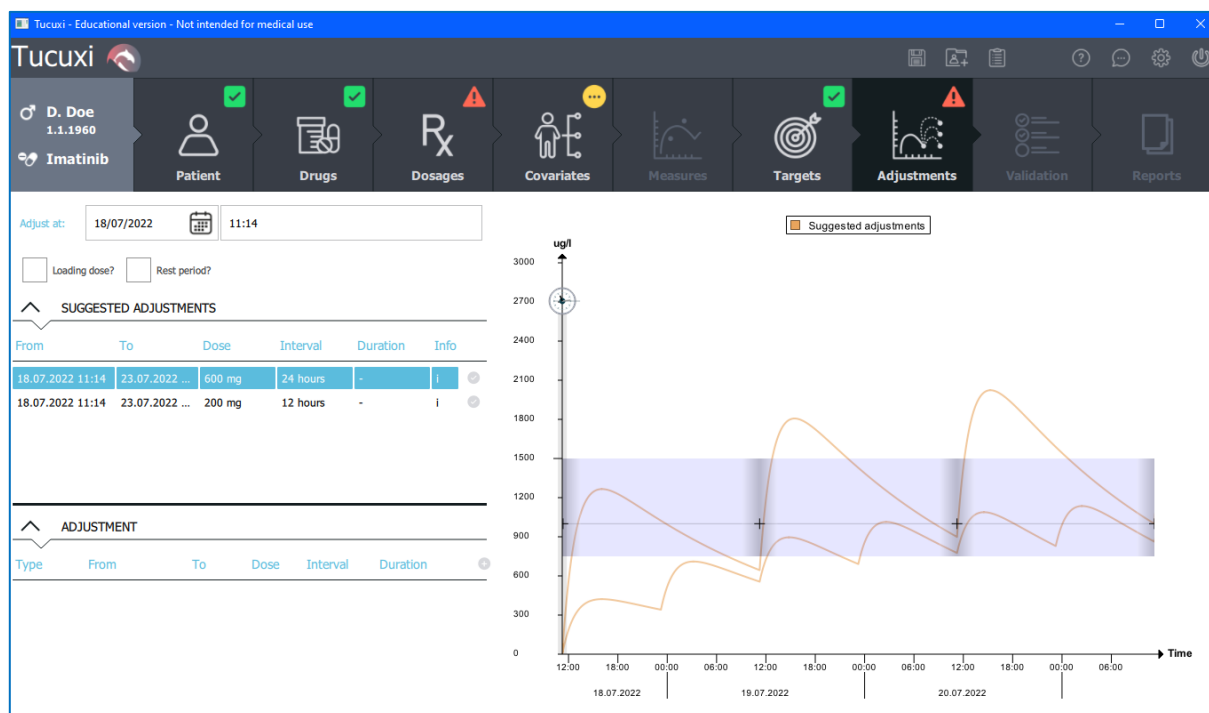


*Figure 2 - Tucuxi graphical user interface with suggested first dosages*

In order to work, Tucuxi needs two types of information:

**Patient information:**

As explained, TDM aims to provide treatments tailored to the patient. In order to best fit the patient's needs, Tucuxi needs as much information as possible such as physical characteristics called covariates, dosages, blood samples and ideal concentrations to be achieved called targets.

**Drug information:**

In order to calculate drug concentration predictions, Tucuxi needs to know the technical characteristics of the drug. All this information is available in what is called a drug model. It indicates which covariates have an influence, formulas to calculate pharmacokinetic parameters, supported formulations and routes of administration, some generic targets and the composition of the drug (what are the active moieties and the analytes).

It is important to know that there is more than one drug model per drug. In general, drug models can differ in almost everything they contain: they can support another set of covariates, another formulation and route of administration, or another set of pharmacokinetic parameters.

The drug model must be chosen according to the information we have about the patient.

## 2.3 Health situation in Tanzania

Tanzania has a high burden of tuberculosis. It is an infectious disease that generally affects the lungs and it is spread through the air by speaking, coughing, spitting or sneezing. According to the world health organization, the TB killed 1.5 million people, it is the second leading infection killer after COVID-19.

Over the last decade, Tanzanian health authorities estimate an incidence of 120'000 – 150'000 patients per year for TB. The global community, through the end of TB strategy, has declared its willingness to end TB by 2035 and a vital component of the arsenal for this includes resorting to the correct use of anti-TB drugs, in particular the first-line agents: isoniazid, rifampicin, ethambutol and pyrazinamide.

However, the problem is that only 60'000 – 75'000 patients are notified and receive a treatment. In addition, studies have reported that rifampicin dosages are insufficient most of the time. For example, investigations by Heysell et al. (2011) and Tostmann et al. (2013) in the Kilimanjaro region showed that one to two thirds of uncomplicated TB patients had maximum concentrations below the reference range of 8-24 mg/L, defined two hours after the last dose intake.

Moreover, a considerable proportion of individuals with TB are co-infected with HIV. It represents 25 to 40% of the monitored people. Semvua et al. (2013) found that the administration of antiretrovirals with first-line anti-tuberculosis drugs further reduces the concentration of rifampicin.

On top of that, TB patients have an increased risk to get affected by diabetes mellitus (DM). It represents 4-16% of the TB population. Unluckily, DM may alter the pharmacokinetics of various drugs, including anti-tuberculosis drugs. Mtabho et al. (2019) observed that DM predicted low levels of rifampicin in TB Tanzanian patients. Sadly, evidence has shown that individuals with TB and DM are five times more likely to die than those without diabetes.

At this point, we easily understand that the risk of treatment failure or unfavorable outcome is real if the dosages stay unsuitable.

# 2.4 Goal of this work

The need to end tuberculosis is real and urgent in Tanzania. Unfortunately, TDM is a long and complicated process, because both the drug measurement and the interpretation of concentration results are demanding. In addition, the number of experienced pharmacologists is not sufficient to provide well-established interpretations and recommendations everywhere their expertise is needed.

Although Tucuxi is an important player in the democratization of TDM, it still needs to be handled by experienced pharmacologists.

This is where TuberXpert comes into play. It comes at the beginning of a large project between Switzerland and Tanzania led by Prof. Thoma Yann, Prof. Mpagama Stellah and Prof. Guidi Monia. The objective is to develop a clinical decision support system (CDSS) to fight tuberculosis . To simplify the development process and reuse what already exists, TuberXpert will be a software that integrates the Tucuxi computation core. By receiving complete information from a patient, the system will assess the relevance of the data provided and then will determine whether a dosage adjustment is necessary or determine the first treatment to be taken. All interpretations made by the program will then be provided to the user in the form of a simplified report compared to the original software. The main purpose of TuberXpert is to simplify the "interpretation and recommendation" phase of TDM for non-experts.



*Figure 3 - TDM process (BUCLIN Thierry, 2022, Les Bases de la pharmacocinétique clinique [PDF document])*

In other words, TuberXpert is a turnkey solution for TDM. The software developed during this Bachelor's thesis is a first step of the whole project. It will then be taken over by three Ph.D. students in charge of the extension and the concrete application of the project.
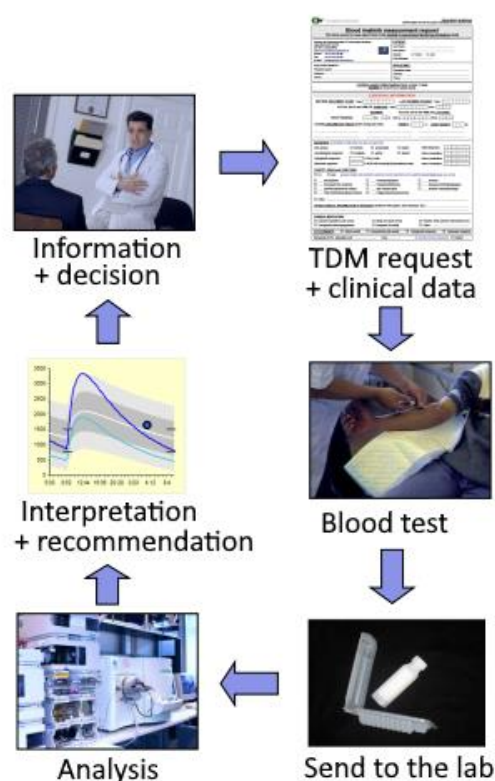
# 3 <u>Work to do</u>

This chapter presents the original specification of this Bachelor's thesis. Then, it describes the requirements: how the system must behave and what are its properties. Finally, some comments on the tests to be performed and the possible architecture to adopt.

## 3.1 Specification

### 3.1.1 Context

This work takes place in the domain of therapeutic drug monitoring (TDM). Currently, Tanzania is experiencing a high amount of tuberculosis cases. The country has expressed its commitment to end TB by 2035. The main tool to achieve this goal is TDM. However, the problem is that it is a long and difficult process. A software called Tucuxi has already been developed and is part of the solution. Nevertheless, it does not solve the fact that the tool needs TDM professionals to be used efficiently. On this perspective, the purpose of this work is to develop a clinical decision support system (CDSS) on top of Tucuxi computation core to automate TDM decision making.

### 3.1.2 Objective

An extensible CDSS must be developed, tested and documented. The CDSS will use a local version of Tucuxi computation core for dosage predictions and treatment adjustment computations. The system will be a command line interface (CLI) that will produce a dosage adjustment report based on the received inputs.

### 3.1.3 Features

**Input validation**
- The program will receive information about the patient and his treatment trough an extensible markup language (XML) file similar to those used with Tucuxi CLI.
    - The XML structure may be extended with new useful elements if needed.
- The program will analyze and verify the relevance of the data.

**Drug file selection**
- The program must be able to select a relevant drug file for each drug in input.

**Dosage adjustment**
- The program must be able to understand the current state of a treatment and suggest an adjustment.

**Output**
- The output of the decisions must be an XML file that can be used by various templates for the report generation.

**Report generation**
- The program must summarize all useful information in a well-formatted report.
    - Suspicious covariates.
    - Drug file selected.
    - Graph "A priori" or "A posteriori", depending on the patient.
    - Dosage adjustments.
    - …

**Multi-language**
- The program must support various languages.
- At least, the English version must be available.
    - It should be easy to add a translation and use it.

**Testing**
- The program behavior must be tested with various inputs.
    - Since it is difficult to predict all cases, obvious cases testing is sufficient.

## 3.2 Requirements

### 3.2.1 Functional

**From the user's point of view**: TuberXpert is an automated clinical decision support system.

It takes as input a query: some information about a patient, his treatments and some requests that provide a few computation instructions. Then, through an output report, the program displays the received data and highlights the suspicious ones, suggests an initial dosage or an adjustment of the drugs and provides any additional useful information for the therapeutic drug monitoring. The output contains readable sentences and the report displays some graphs of the adjustments.

**From the developer's point of view**: TuberXpert is an additional software layer on top of Tucuxi computation core.

As input, TuberXpert needs an XML file containing patient and treatments information as well as requests that contain computation instructions. The structure of the file is the same as that of Tucuxi CLI. However, it could extend the basic structure by adding some new elements, at least, a new type of request dedicated to TuberXpert control and probably some administrative information.

The first step in the execution is to obtain a relevant drug file. TuberXpert does not ask the user to specify which drug model to use. It chooses the most appropriate one based on the available patient information.

Then, the program performs a validity check on the data provided. It assesses whether the values received are plausible. Are they normal? Did the practitioner make a typing error?

After the validation, TuberXpert prepares an adjustment request that it submits to Tucuxi computation core.

Finally, it generates an output report corresponding to a format requested by the user. This can be XML, HTML or PDF. The report contains some full sentences, the evaluation of the data, the dosage adjustments graphs for the HTML and PDF format or the adjustment data to generate a graph for the XML format. The report can be generated in several languages.

### 3.2.2 Nonfunctional

This project is most likely a proof of concept. For this work, it is not necessary to develop a graphical user interface. The program will be run as a CLI.

As long as it is relevant, the generated report must be good-looking and user-friendly. In other words, it should not be painful to read and to extract the information.

Although this work is made in the context of tuberculosis, it is developed without an extensive knowledge of rifampicin or any other drug. Consequently, the development is generic in considering all drugs. Therefore, it should be easy to edit the parts specific to the adaptation of a drug.

## 3.3 Testing

The program behavior must be evaluated with various inputs. Since it is difficult to predict all cases, obvious cases testing is enough.

## 3.4 Architecture

In terms of software architecture, the CDSS may be separated from the report generation. The CDSS must offer a standardized output, most probably in XML format. This output may be used by a third-party report generator to fill some fields of a report template.

# 4  Analysis

This chapter presents the analysis that is done prior to the implementation part. The objective is to anticipate and take some decisions based on business and technical analysis.

## 4.1 Technologies

### 4.1.1  Development language

Initially, Tucuxi was developed in C++. This language was chosen for its superior performance since the software requires a lot of computing power. Then, other projects were added. Most of them were also developed in C++ for the same reasons.

Thus, to preserve the homogeneity of the collection of projects, TuberXpert will also be developed in C++. This language is once again very advantageous. As the software could be used on low-powered computers in Tanzania, the performance will be optimized by using a low-level language.

Version: C++ 17

### 4.1.2  Integrated Development Environment

Once again, it will walk in the steps of Tucuxi, and it will use the same development environment. Qt Creator is a very advantageous and rich integrated development environment (IDE). It easily allows cross-platform development, tests management, language management and many other frameworks. Qt Creator is very versatile. It also makes it easier to work with existing projects at the same time. De facto, it is the IDE that will be used to develop TuberXpert.

Version: Qt Creator 6.0.2 community

### 4.1.3  Compilers

As this project should be easy to use on as many computers as possible, I have chosen two classic and widely used compilers.

Versions:

- Windows: MinGW-W64 6.3.0
- Ubuntu: GCC 11.2.0

## 4.2 Global application overview

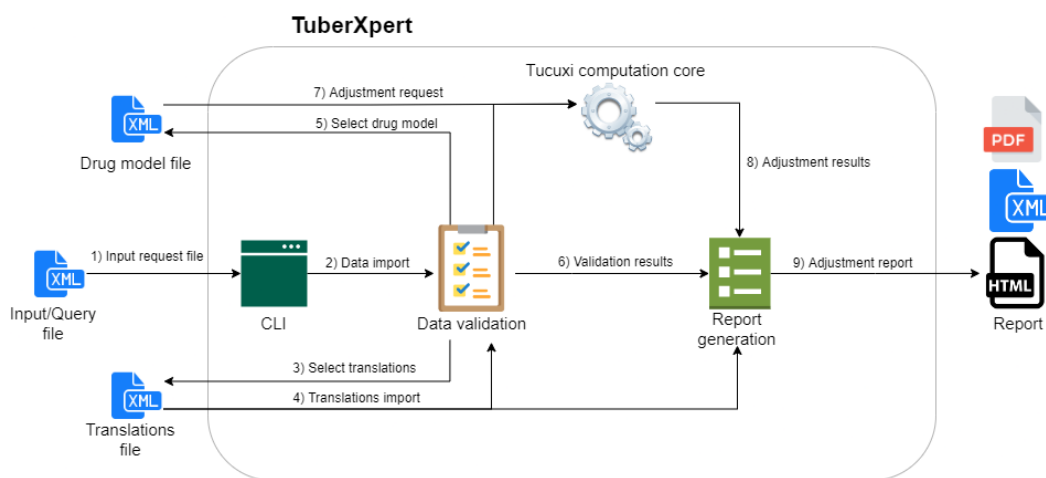This diagram shows what will be included in TuberXpert and what is the order of execution.



*Figure 4 - Global application overview and components*

1. The program receives as input the patient and his treatments information in XML format. The file structure is explained in the next section.
2. The program loads the data from the query file.

This is where the validation phase begins. The following steps are repeated for each TuberXpert request in the query file. This allows each adjustment to be processed independently ensuring that some that fail do not impact others that might succeed (e.g., a missing translations file).

3. The TuberXpert specific request is analyzed and determines the translations file to be loaded according to the required output language.
4. The translations file is loaded.
5. A drug model is selected that matches the patient information.
6. Data validation is complete and the results are ready for the report generation.
7. Based on the data validation, an adjustment request is made for Tucuxi computation core.
8. The adjustment data are provided to the report generator.
9. The adjustment report is generated according to the required output format.

I chose to include the report generation engine directly in TuberXpert to facilitate the use and to have a homogeneity of the project. However, thanks to the XML export, it is always possible to have an external report generator if the integrated one is not sufficient.

## 4.3 Input – TuberXpert query file

Like the Tucuxi CLI, TuberXpert receives all the useful information about the patient, his treatments and the adjustments to be performed through an XML query file. Most of its structure will be essentially the same as Tucuxi CLI query.

> *For a complete Tucuxi CLI query specification, see "tiersdoc/Tucuxi_CLI_Usability_Specification.pdf".*

Although, this section reviews the TuberXpert query structure, it does not explain how to form the common elements with Tucuxi CLI but what they represent and, if necessary, what will be checked. When a relevant point is reached, it explains what additional element needs to be added to meet the needs of TuberXpert, such as administrative data and the TuberXpert requests.

## 4.3.1 Global

Down below is the overall structure of the input. It consists of the date of computation and the administrative data (admin). Then, there is information about the patient's covariates followed by the drugs he is taking. Finally, comes the requests element. It contains the new xpertRequest elements used to tell TuberXpert which drug should be adjusted and how the adjustment report should be.

**Example of the global structure of the query**:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<query version="1.0"
    xmlns:xsi="http://www.w3.org/2001/xmlSchema-instance"
    xsi:noNamespaceSchemaLocation="computing_query.xsd">


    <date>2018-07-11T13:45:30</date>


    <admin>[…]</admin>


    <drugTreatment>
        <patient>
            <covariates>[…]</covariates>
        </patient>
        <drugs>[…]</drugs>
    </drugTreatment>
    <requests>
      <xpertRequest>[…]</xpertRequest>
      […]
    </requests>
</query>
```

> *In the Tucuxi CLI query, there are the queryId, clientId and language elements. In TuberXpert, these elements are not necessary. So, they are not necessarily present. Although their value is currently unused, they are still retrieved if they are present.*

The date will be used to fix the "present" time. It will be particularly useful to calculate an age from a date of birth in a test and getting the same result today and in ten years. However, the general practice will be to put the local time in this element.

The noticeable addition is the admin element and the xpertRequest elements.

## 4.3.2 Admin

The structure of the admin element is inspired by the one from the Bachelor's thesis of Nadir Benallal. It is flexible and contains every needed field to store contact information of the patient and the mandator as well as clinical data of any kind.

This administrative information will be used to find out who the patient is, who the adjustment mandator is and how they can be contacted. It should be displayed at the beginning of the report to know who is involved.

**Checks:**

At this stage, no data validation is performed.

```
<admin>
    <mandator>
        <person>[…]</person>
        <institute>[…]</institute>
    </mandator>
    <patient>
        <person>[…]</person>
        <institute>[…]</institute>
     </patient>
     <clinicalData>[…]</clinicalData>
</admin>
```

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| <admin> | | 0:1 | The administrative data. |
| _<mandator> | | 0:1 | The mandator of the adjustment. |
| __<person> | Person | 1:1 | Personal contact information of the mandator. |
| __<institute> | Institute | 0:1 | Institute contact information of the mandator. |
| _<patient> | | 0:1 | The patient that will follow the adjustment. |
| __<person> | Person | 1:1 | Personal contact information of the patient. |
| __<institute> | Institute | 0:1 | Institute contact information of the patient. |
| _<clinicalDatas> | | 0:1 | Contains the clinical data. |
| __<clinicalData> | string | 0: ∞ | Any additional data. |

The *clinicalData* element has an attribute called *"key"*. The content of this attribute is used as key to retrieve the value. For example: *<clinicalData key="roomNumber">25</clinicalData>*

It is recommended to write the *key* value in camel case so that it will be correctly rendered in the HTML/PDF report.

**Person element**

A person element contains the personal contact information of the patient or the mandator.

```
<person>
    <id>XXX.XXXX.XXXX.XX</id>
    <title>Mr.</title>
    <firstName>John</firstName>
    <lastName>Doe</lastName>
    <address>[…]</address>
    <phone>[…]</phone>
    <email>[…]</email>
</person>
```

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| <person> | | 1:1 | Personal contact information. |
| _<id> | string | 0:1 | Identifier of the person. |
| _<title> | string | 0:1 | Title of the person. |
| _<firstName> | string | 1:1 | The first name of the person. |
| _<lastName> | string | 1:1 | The last name of the person. |
| _<address> | Address | 0:1 | Address of the person. |
| _<phone> | Phone | 0:1 | Phone number of the person. |
| _<email> | Email | 0:1 | Email of the person. |

## Institute element

An institute element contains the contact information of an institute.

```
<institute>
    <id>456789</id>
    <name>CHUV</title>
    <address>[…]</address>
    <phone>[…]</phone>
    <email>[…]</email>
</institute>
```

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| <institute> | | 0:1 | Institute contact information. |
| _<id> | string | 0:1 | Identifier of the institute. |
| _<name> | string | 1:1 | Name of the institute. |
| _<address> | Address | 0:1 | Address of the institute. |
| _<phone> | Phone | 0:1 | Phone number of the institute. |
| _<email> | Email | 0:1 | Email of the institute. |

## Address element

An address element contains the address information of a person or an institute.

```
<address>
    <street>Av. de l'Ours 1</street>
    <postalCode>1010</postalCode>
    <city>Lausanne</city>
    <state>Vaud</state>
    <country>Suisse</country>
</address>
```

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| <address> | | 0:1 | Address of a person or an institute. |
| _<street> | string | 1:1 | Street of the address. |
| _<postalCode> | string | 1:1 | Postal code of the address. |
| _<city> | string | 1:1 | City of the address. |
| _<state> | string | 0:1 | State of the address. |
| _<country> | string | 0:1 | Country of the address. |

**Phone element**

A phone element contains a number and a type of phone number for a person or an institute.

```
<phone>
      <number>0213140001</number>
      <type>private</type>
</phone>
```

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| <phone> | | 0:1 | Phone number of a person or an institute. |
| _<number> | string | 1:1 | Phone number. |
| _<type> | string | 0:1 | Phone type. |

> *The <type> tag is a string enumeration. It can be "private" or "professional".*

**Email element**

An email element contains an email address and its type for a person or an institute.

```
<email>
    <address>anemail@email.mail</address>
    <type>professional</type>
</email>
```

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| <email> | | 0:1 | Email of a person or an institute. |
| _<address> | string | 1:1 | Email address. |
| _<type> | string | 0:1 | Email type. |

> *The <type> tag is a string enumeration. It can be "private" or "professional".*

### 4.3.3 Covariates

The covariates element contains the list of the patient's known covariates defined by an identifier (covariateId), a date of measure, a value, a value type (dataType) and a unit.

```
<covariates>
    <covariate>
        <covariateId>bodyweight</covariateId>
        <date>2018-07-11T10:45:30</date>
        <value>70</value>
        <unit>kg</unit>
        <dataType>double</dataType>
    </covariate>
    […]
</covariates>
```

**Checks:**

The covariate value is checked using the covariate definition validation of the drug model file.

> *In drug models, each required covariate has a covariate definition. It provides a default value and a validation formula that indicates whether the corresponding patient covariate value is suitable or not. If this validation domain is not met, the drug model may not fit. If the value of a covariate is not expected, the influenced pharmacokinetic parameters may have unverified values that may lead to implausible predictions.*
>
> *After a drug file is selected, each patient covariate that does not meet the validation domain of the same covariate definition in the drug file will generate a warning in the final report.*

### 4.3.4 Drugs

The drugs element contains some drug elements. They represent the treatments the patient is undergoing. A drug element typically contains the associated drug identifier (drugId), the name of the active principle (activePrinciple), the manufacturer's brand name (brandName), the drug atc, the patient's treatment, the patient's blood samples and the targets the patient must reach.

```
<drugs>
    <drug>
        <drugId>rifampicin</drugId>
        <activePrinciple>something</activePrinciple>
        <brandName>somebrand</brandName>
        <atc>something</atc>
        <treatment>[…]</treatment>
        <samples>[…]</samples>
        <targets>[…]</targets>
    </drug>
    […]
</drugs>
```

**Checks:**

The identifier of the drug for which an adjustment is requested must match at least one drug model file.

> *If there are no matching drug model files, the adjustment for that drug will be dropped.*

## 4.3.5 Treatment

The dosages are in the treatment element. It contains the patient's dosage history for a given drug. A dosage has a start date and an end date. It shows what the patient takes, when and on what basis.

The dosage element is complex but flexible. It allows describing dosages such as "take a drug at 8:00 every day except on Sunday." The main point is that it will always contain a dose element that allows the dosage validation.

```
<treatment>
    <dosageHistory>
        <dosageTimeRange>
            <start>2018-07-06T08:00:00</start>
            <end>2018-07-08T08:00:00</end>
            <dosage>
                […]
                <dose>
                    <value>400</value>
                    <unit>mg</unit>
                    <infusionTimeInMinutes>60</infusionTimeInMinutes>
                </dose>
                […]
            </dosage>
        </dosageTimeRange>
    </dosageHistory>
    […]
</treatment>
```

**Checks:**

Each dose will be converted to match the unit of the available doses of the drug model file that will be selected. Then, each value will be compared to the domain of the available doses from the drug model file.

> *If a dose reaches the minimum or maximum bounds from the drug file, a warning will be printed in the final report.*

## 4.3.6 Samples

The samples element contains the patient's blood samples. In other words, a list of drug concentration measurements. A sample is defined by an identifier (sampleId), a date of measure (sampleDate) and some concentrations. There are multiple concentrations when the drug contains multiple analytes. Therefore, a concentration contains its associated analyte identifier (analyteId), a value and a unit.

```
<samples>
    <sample>
        <sampleId>1</sampleId>
        <sampleDate>2018-07-07T06:00:00</sampleDate>
        <concentrations>
            <concentration>
                <analyteId>imatinib</analyteId>
                <value>0.7</value>
                <unit>mg/l</unit>
            </concentration>
        </concentrations>
    </sample>
    […]
</samples>
```

**Checks:**

In order to check the samples, the program will compute an "A priori" estimation, i.e., with the patient's covariates instead of typical patient characteristics . Then, it will check if the samples are below or above certain percentiles.

> *If a sample is below the percentile X or above the percentile Y, the program will print a warning message in the final report.*

## 4.3.7 Targets

The adaptation engine uses targets to adapt the dosage to the patient's needs. The drug model files provide such targets, but they correspond to the typical patient. In other words, those targets are generic and not patient specific. Therefore, it is possible to replace them by providing some in the query file. A target contains a corresponding active moiety (activeMoietyId), a type (targetType), a unit and some thresholds: minimum to reach (min), maximum to reach (max), best to reach, inefficiency limit (inefficacyAlarm) and toxicity limit (toxicityAlarm).

```
<targets>
    <target>
        <activeMoietyId>imatinib</activeMoietyId>
        <targetType>residual</targetType>
        <unit>mg/l</unit>
        <min>20</min>
        <best>25</best>
        <max>30</max>
        <inefficacyAlarm>15</inefficacyAlarm>
        <toxicityAlarm>50</toxicityAlarm>
    </target>
    […]
</targets>
```

**Checks:**

For a personalized target to be valid, it must have the same active moiety identifier as the drug model file ones, but it must not have an identical active moiety identifier and target type to another personalized target.

> *If a target is invalid, the treatment adjustment will abort and return a specific error.*

## 4.3.8  TuberXpert requests

The requests element contains some xpertRequest elements which is the way the user tells TuberXpert what to do. This is where the last change from the original query format occurs.

```
<requests>
    […]
    <requestXpert>[…]</requestXpert>
    […]
</requests>
```

> *In the Tucuxi CLI query, the requests element contains some "request" elements. It is the user's way of controlling the Tucuxi computation core. It is a highly configurable element that tells the core whether it should calculate a prediction, an adjustment or some percentiles. TuberXpert does not need this element because the computing instructions for Tucuxi computation core will be created by TuberXpert itself. Consequently, this element is not necessary. So, it is not necessarily present. Although its value is currently unused, it is still parsed if it is present.*

An xpertRequest contains two types of information:

How to generate the report?

- In English? In French?
- XML? HTML? PDF?

What drug to adjust and how?

- The identifier of the drug to be adjusted.
- If we know when, the date of adjustment.

And for more advanced users:

- Is a loading dose or a rest period allowed to reach the target faster at the beginning of the treatment?
- What type of target to use? Should we use the ones from the query or the ones from the drug model? What to do when there are both?
- What formulation and administration route should be used for the adjusted treatment?

**Checks:**

No particular check is performed.

```
<xpertRequest>
    <drugId>rifampicin</drugId>
    <output>
        <format>xml</format>
        <language>en</language>
    </output>
    <adjustmentDate>2022-07-06T08:00:00</adjustmentDate>
    <options>
        <loadingOption>noLoadingDose</loadingOption>
        <restPeriodOption>noRestPeriod</restPeriodOption>
        <targetExtractionOption>populationValues</targetExtractionOption>
        <formulationAndRouteSelectionOption>lastFormulationAndRoute
                                        </formulationAndRouteSelectionOption>
    </options>
</xpertRequest>
```

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| <xpertRequest> | | 1: ∞ | The request for TuberXpert. |
| _<drugId> | string | 1:1 | The identifier of the drug to adjust. |
| _<output> | | 1:1 | Specification about the output. |
| __<format> | string | 1:1 | Output format. |
| __<language> | string | 1:1 | Output language. |
| _<adjustmentDate> | date | 0:1 | Date of adjustment. |
| _<option> | | 0:1 | Options specification. |
| __<loadingOption> | string | 0:1 | Allow a loading dose or not. |
| __<restPeriodOption> | string | 0:1 | Allow a rest period or not. |
| __<targetExtractionOption> | string | 0:1 | Extraction option for targets. |
| __<formulationAndRouteSelectionOption> | string | 0:1 | Selection of the potential formulation and administration route. |

As we can see, TuberXpert can work without any configuration, except for the name of the drug to process and the format of the report.

> *The <format> tag is a string enumeration that allows choosing the output format. It can be "html", "xml" or "pdf".*

> *The <language> tag is a string enumeration that allows choosing the output language. It can be "en" or "fr".*

The <loadingOption> tag is a string enumeration. It can be "noLoadingDose" or "loadingDoseAllowed".

From Tucuxi CLI Software Usability Specification:

- "noLoadingDose: No loading dose can be added to the new dosage"
- "loadingDoseAllowed: If the current dosage is under the target, a loading dose can be added at the beginning of the new dosage to more rapidly reach the optimum."

If the tag is not present, the recommendation from the drug model is used.

---

The <restPeriodOption> tag is a string enumeration. It can be "noRestPeriod" or "restPeriodAllowed".

From Tucuxi CLI Software Usability Specification:

- "noRestPeriod: No rest period can be added to the new dosage"
- "restPeriodAllowed: If the current dosage is over the target, a rest period can be added at the beginning of the new dosage to more rapidly reach the optimum."

If the tag is not present, the recommendation from the drug model is used.

---

The <targetExtractionOption> tag is a string enumeration. It can be "populationValues", "aprioriValues", "individualTargets", "individualTargetsIfDefinitionExists", "definitionIfNoIndividualTarget" or "individualTargetsIfDefinitionExistsAndDefinitionIfNoIndividualTarget".

From Tucuxi CLI Software Usability Specification:

- "populationValues: Forces the population values to be used"
- "aprioriValues: Forces the a priori values to be calculated and used"
- "individualTargets: Only use the individual targets"
- "individualTargetsIfDefinitionExists: Only use the individual targets if a target definition exists"
- "definitionIfNoIndividualTarget Use the individual target, and if for an active moiety and a target type no individual target exists, then use the definition"
- "individualTargetsIfDefinitionExistsAndDefinitionIfNoIndividualTarget: Use the individual target if a target definition exists, and if for an active moiety and a target type no individual target exists, then use the definition"

If the tag is not present, the value "definitionIfNoIndividualTarget" is used.

---

The <formulationAndRouteSelectionOption> tag is a string enumeration. It can be "lastFormulationAndRoute", "defaultFormulationAndRoute" or "allFormulationAndRoutes".

From Tucuxi CLI Software Usability Specification:

- "lastFormulationAndRoute: Use only the last formulation and route used in the current treatment. If the treatment is empty, then use the default formulation and route of the drug model."
- "defaultFormulationAndRoute: Use only the default formulation and route of the drug model"
- "allFormulationAndRoutes: Use all available formulation and routes of the drug model"

If the tag is not present, the value "lastFormulationAndRoute" is used.

## 4.4 Program execution flow

This chapter presents the execution flow steps when running the program. The colors represent the main steps that are detailed in specific subchapters.
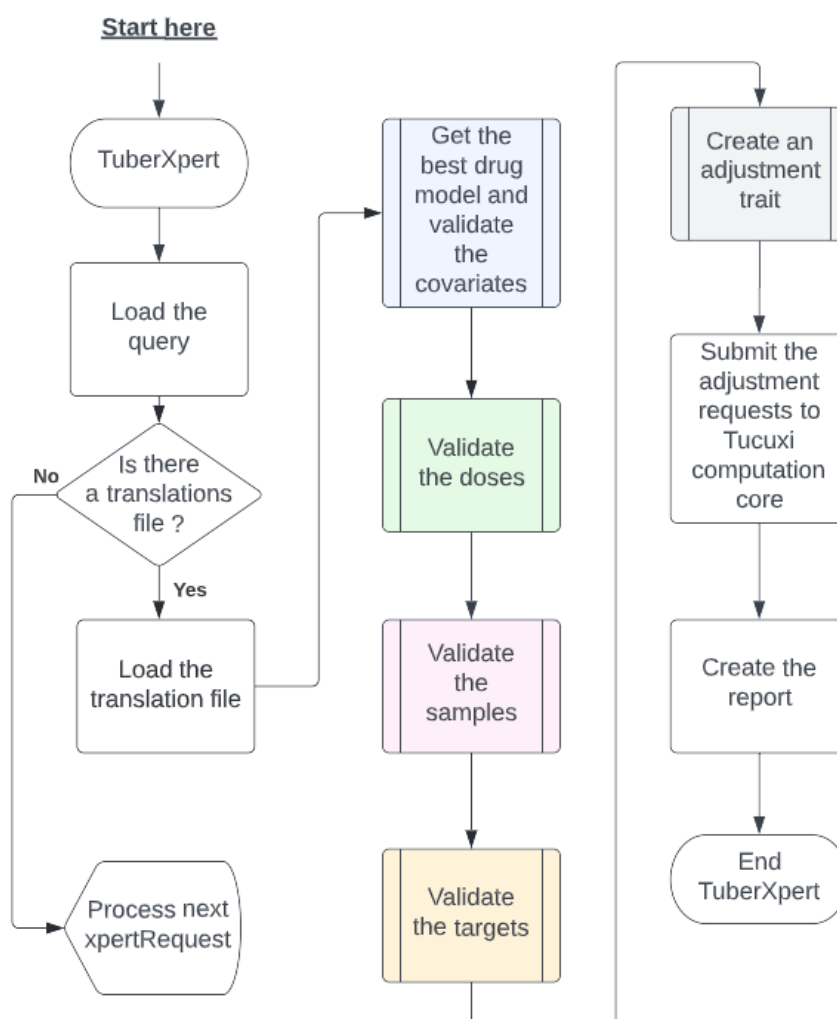


*Figure 5 - Program global execution*

The first step is to load the query for processing. This is the only step performed once. The next flow steps are done for each TuberXpert request:

- 1st flow step) Check if there is a translations file for the required output language and load it.
- 2nd flow step) Try to get the best drug model and to validate the covariates.
- 3rd flow step) Try to validate the doses.
- 4th flow step) Try to validate the samples.
- 5th flow step) Try to validate the targets.
- 6th flow step) Try to create the adjustment trait used by Tucuxi computation core to calculate an adjustment.
- 7th flow step) Based on the trait from the last step, we are able to perform the adjustment computations to obtain: the adjustment data, the pharmacokinetic parameters of different types ("Typical patient", "A priori" and eventually "A posteriori") and extrapolated steady state statistics.
- 8th flow step) Try to print the report.

Any of these steps may fail. In this case, the TuberXpert request being processed is abandoned and the next request starts to be processed.

## 4.4.1 Get the best drug model and validate the covariates

One task of the system is to choose a drug model file to use. Since the drug model selection requires knowing which covariates are present and valid, we take the opportunity to do covariate validation at the same time.
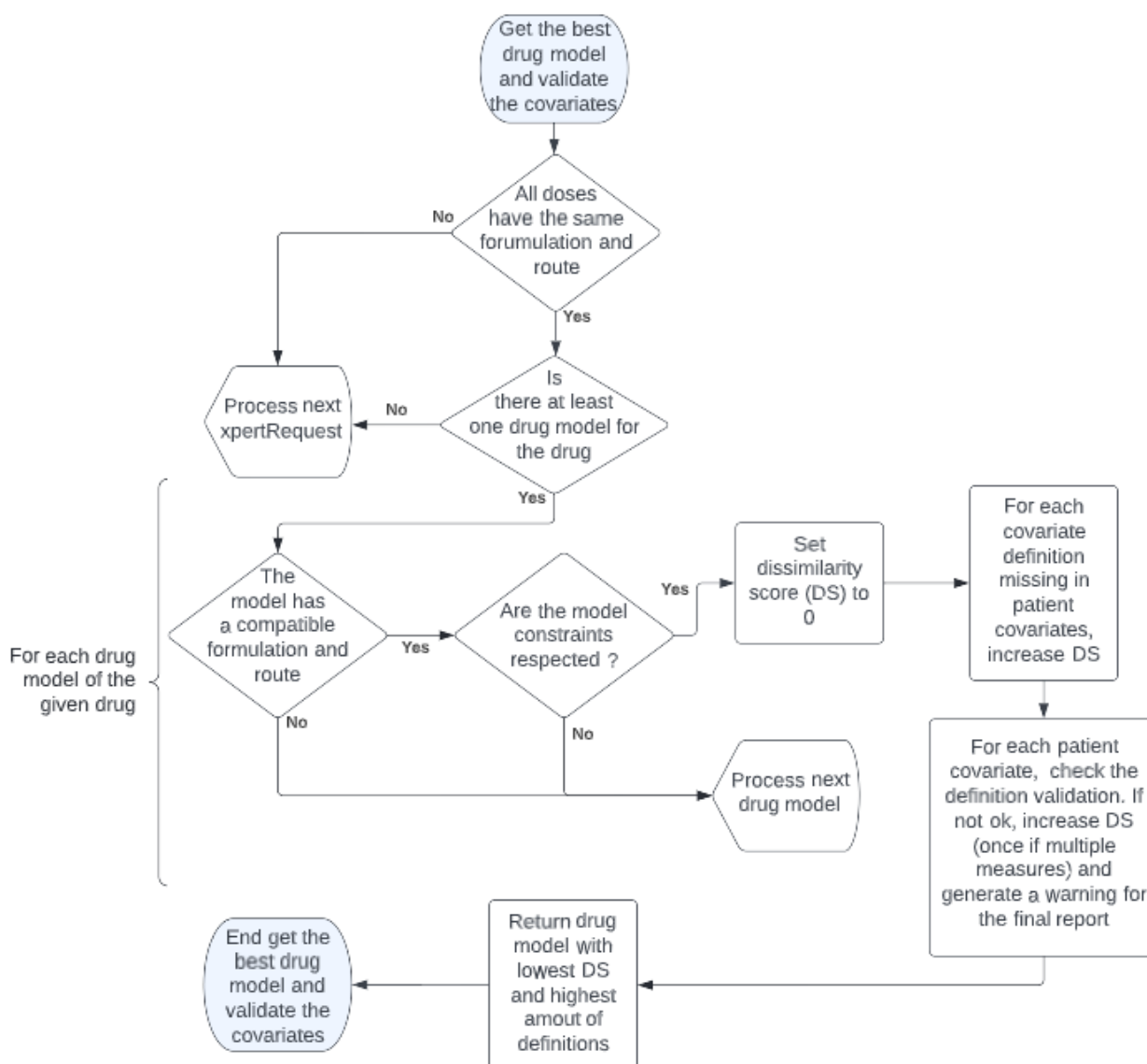


*Figure 6 - Process of drug model selection and covariate validation*

Firstly, we will verify that all the doses of the patient's treatment have the same formulation and administration route. Then, for each drug model available for the drug of interest, we check that the formulation and route of administration of the treatment are supported, otherwise the model is dropped. After that, we check that the constraints are respected. A drug model can say "If you are a woman, your body weight must be bigger than 40 kg". In other words, constraints are preliminary conditions that define if we can use the model or not. Next, we calculate a dissimilarity score (DS) based on the model covariate definitions.

$$DS = \sum missing\ covariate\ definitions\ in\ treatment\ + \\ \sum covariates\ of\ the\ treatment\ not\ respecting\ the\ definitions\ (once\ per\ covariate\ id.)$$

The model with the lowest dissimilarity score is chosen. In case of a tie, the model with the most covariate definitions is chosen. The method is not optimal, but it is a good starting point. What happens if two models tie perfectly, but one may be fitting better? This type of question is not considered by this algorithm. In the future, a close collaboration with pharmacologists would be necessary to determine for each drug, "how to effectively choose the drug model file that fits the most to the patient for a given drug".

*Drug model constraints and covariate definition validations do almost the same thing but have very different effects:*

- *A constraint has the power to say that the drug model should not be used while the covariate validation simply says whether the covariate value is expected or not, without much consequence.*
- *A constraint can consider several covariates whereas a covariate definition is more likely to consider only the value of the covariate to which it is related.*

*At this point, we can say:*

*If the input doses have the same formulation and administration route:*

- *Yes: Search for the best drug model.*
- *No: Return an error for this TuberXpert request and process the next one.*

*Is there a drug model that matches the patient's information?*

- *No: Return an error for this TuberXpert request and process the next one.*

*In regards of the selected drug model, are some covariate definitions missing from the patient's covariates?*

- *Yes: Use their default values.*

*In regards of the selected drug model, are the patient's covariate values supported by the covariate definitions of the drug model?*

- *No: Generate a warning for those covariates in the report. Maybe a double check is required.*

## 4.4.2  Validate the doses

To adjust a treatment, it is important that the doses used to perform the adjustment computation are relevant. Therefore, after loading the best drug model, the CDSS will check that every dose of the request matches the recommended dose range from the drug model.
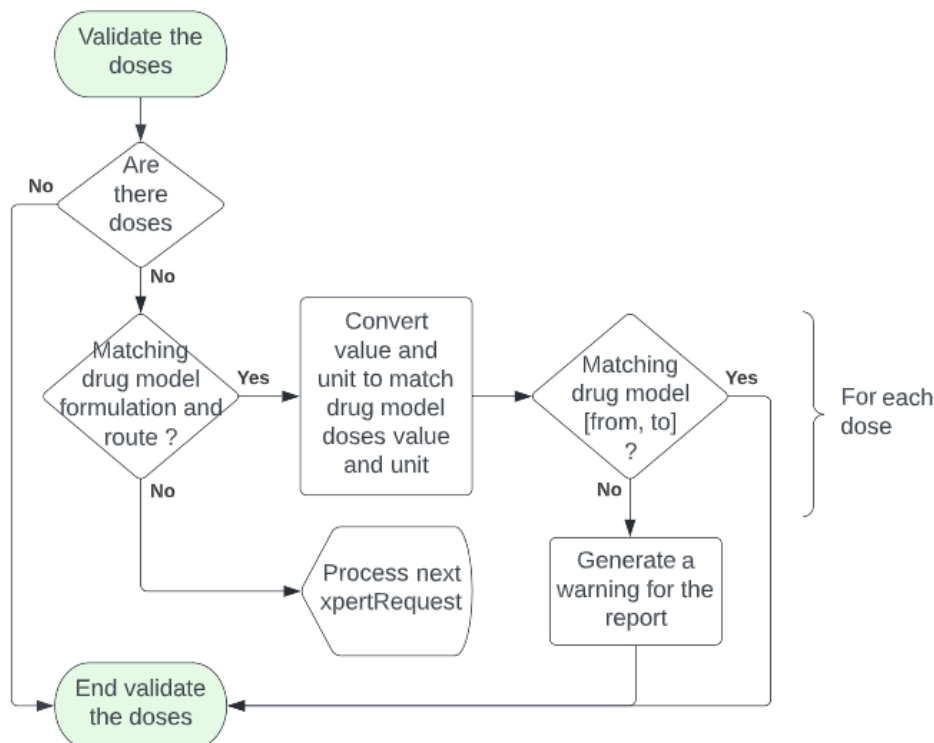


*Figure 7 - Process of dose validation*

Every drug model file has an "availableDoses" element that has a unit and a range [from, to]. This information is used to validate the doses of the treatment. The Tucuxi computation core unit manager can convert a value from a base unit into a target unit. Thus, the CDSS will be able to convert each dose and verify that they are within the range of the formulation and route of administration of the corresponding drug model file.

> *At this point, we can say, if the doses are normal, i.e., within the normal range:*
>
> - *No: Generate a warning in the report. Maybe a double check is required.*

## 4.4.3 Validate the samples

The samples need to be representative of what was really measured. If the samples are wrong, the computation core will compute a wrong adjustment because the samples are not representative of the response of the body to the treatment.
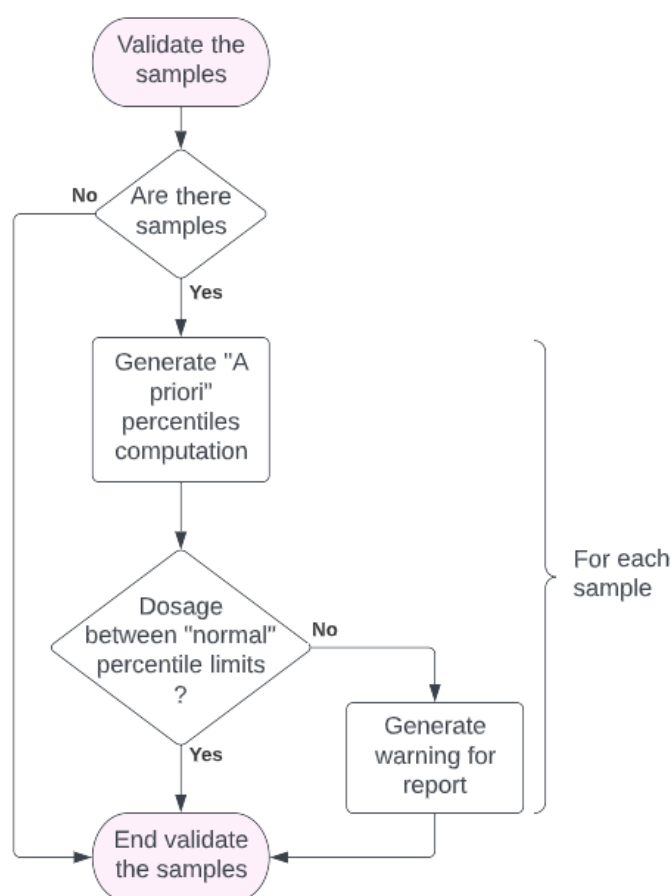


*Figure 8 - Process of sample validation*

Using Tucuxi computation core, the system will generate a percentile prediction based on the patient's covariates. Considering the patient's covariates, it will be possible to determine which percentiles the patient is in.

For example, we will use 4 percentiles that will change the level of warning:

- Below 5 or above 95: a critical warning.
- Below 10 or above 90: a normal warning.

> *At this point, we can say, if the samples are normal, i.e., above and below a certain percentile:*
>
> - *Yes: No problem.*
> - *No: Generate a warning in the report. Maybe a double check is required.*

## 4.4.4 Validate the targets

In the input XML file, it is possible to create custom targets that override the default targets of the drug model file.
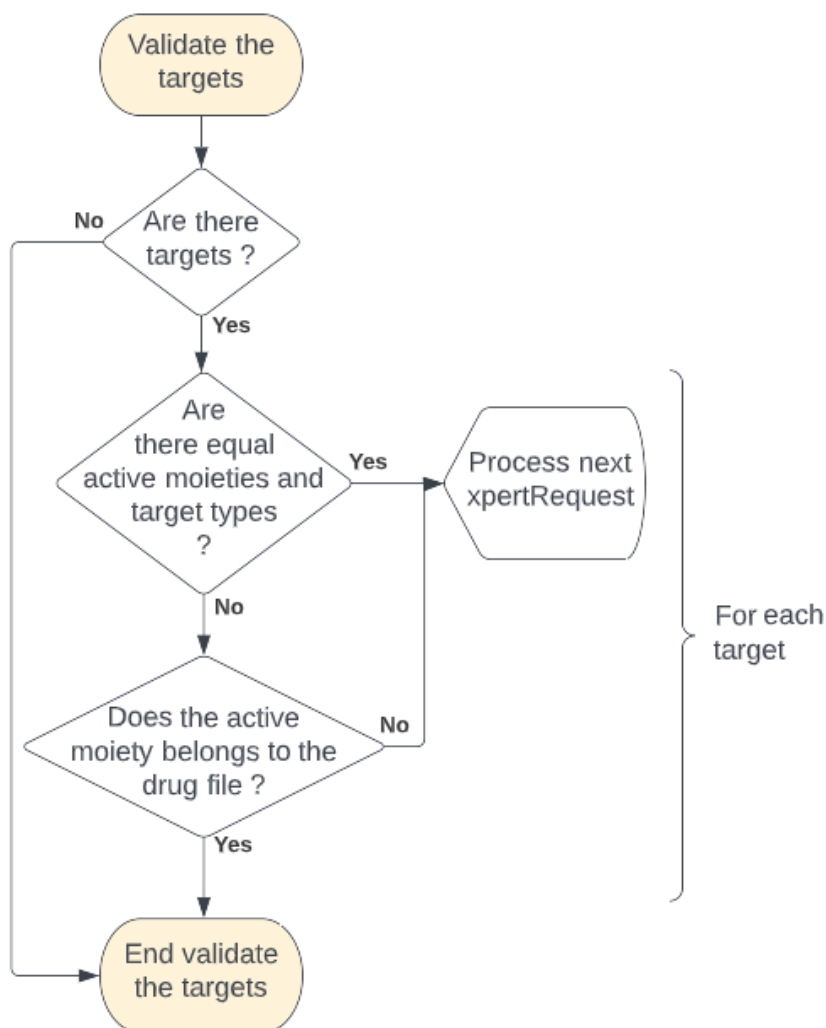


*Figure 9 - Process of target validation*

Firstly, it checks if two custom targets have the same active moiety and the same target type. In this case, the targets are redundant. Since we cannot choose between these targets, we display an error and stop the adjustment for the relevant drug. Then, it checks that the active moiety of the custom target is an active moiety of the drug model file.

> At this point, we can say, if the custom targets are normal, i.e., non-redundant and using a good active moiety of the related drug model:
>
> - Yes: Keep them.
> - No: Return an error for this TuberXpert request and process the next one.

> The possibility to write custom targets is normally intended for experienced practitioners.
>
> There is no absolute rule that is easy to implement to determine what constitutes a relevant target for every drug. As a result, this version of the CDSS does not check for unit, min, max, best, and alarms values.
>
> In future versions, specific rules should be implemented for each drug.

## 4.4.5 Create adjustment trait

The last step before launching an adjustment computation is to take the last decisions to prepare the adjustment trait that will be used by Tucuxi computation core to perform the treatment adjustment.
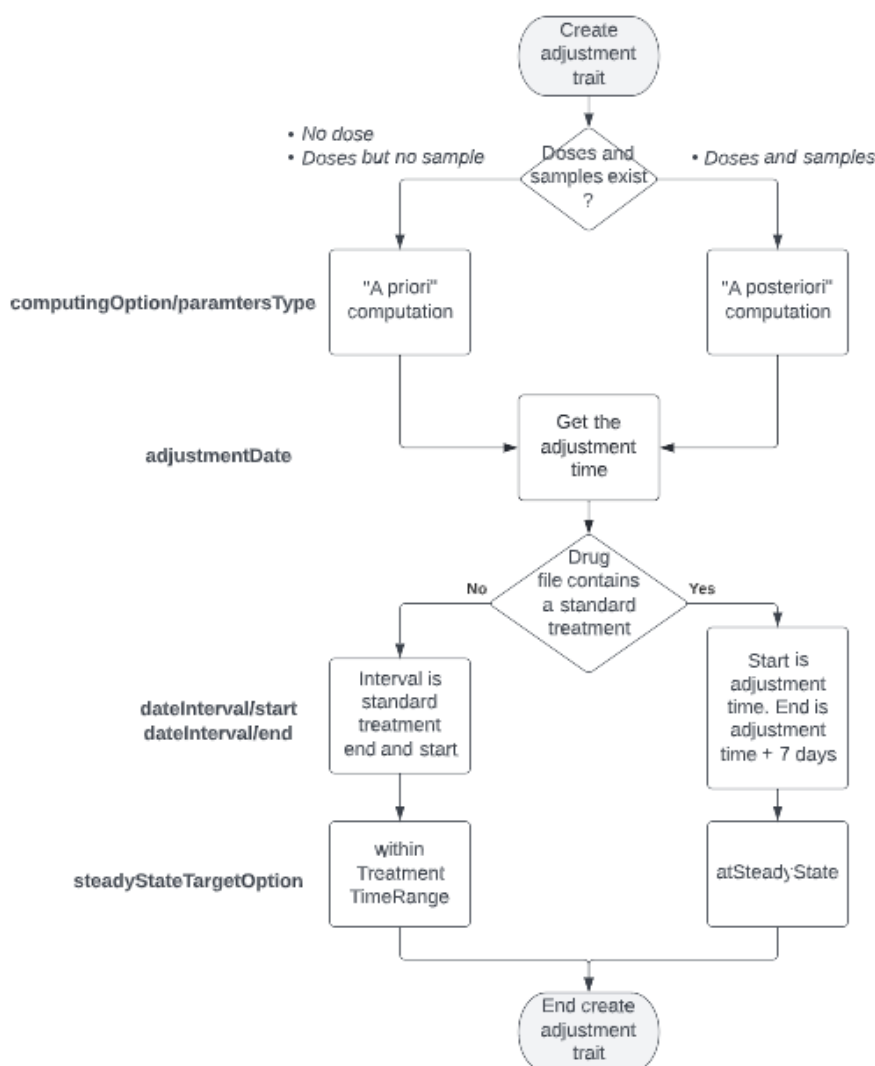


*Figure 10 - Process of adjustment trait creation*

At that point, we need four more information to create the adjustment trait:

- What is the pharmacokinetic parameters type?
  - If the patient does not have any dosage or sample, the parameters type is "A priori".
  - If the patient has dosages and samples, the parameters type is "A posteriori".

- When is the treatment adjustment?
  - If the value is set in the TuberXpert request, use it.
  - Otherwise
    - If there is a treatment in progress, use the next intake time.
    - If there is a completed treatment, from the last intake time, add "2 * half-life of the drug" until the computation time is reached. The resulting time used[1].
    - If there is no treatment, use the computation time plus 1 hour.

---

[1] The half-life is the time it takes for the amount of active substance in a drug to decrease by half. It is in the drug model.

- What is the adjustment interval for which we want the data?
    - If the drug model does not specify a standard treatment, the interval starts at the adjustment time and ends 7 days after the start date.
    - If the drug model specifies a standard treatment, the interval starts and ends accordingly to the standard treatment interval and the first intake.

- What is the steady state target option?
    - If the drug model does not specify a standard treatment, the option is *atSteadyState.*
    - If the drug model specifies a standard treatment, the option is *withinTreatmentTimeRange.*

Once the program gets these answers, it will be able to create a Tucuxi request with this adjustment trait. Some elements are forced by TuberXpert, but some others could be tweaked by the user using the TuberXpert custom request.

For example, here is a complete adjustment request for Tucuxi computation core (in TuberXpert, we will only use the equivalent C++ object):

```xml
<request>
    <requestId>adjustment</requestId>
    <drugId>rifampicin</drugId>
    <drugModelId>ch.tucuxi.rifampicin.svensson2017.tdd</drugModelId>
    <adjustmentTraits>
        <computingOption>
            <parametersType>aposteriori</parametersType>
            <compartmentOption>allActiveMoieties</compartmentOption>
            <retrieveStatistics>true</retrieveStatistics>
            <retrieveParameters>true</retrieveParameters>
            <retrieveCovariates>true</retrieveCovariates>
        </computingOption>
        <nbPointsPerHour>20</nbPointsPerHour>
        <dateInterval>
            <start>2018-01-12T07:00:00</start>
            <end>2018-03-15T12:59:00</end>
        </dateInterval>
        <adjustmentDate>2018-01-12T07:00:00</adjustmentDate>
        <options>
            <bestCandidatesOption>bestDosagePerInterval</bestCandidatesOption>
            <loadingOption>noLoadingDose</loadingOption>
            <restPeriodOption>noRestPeriod</restPeriodOption>
            <steadyStateTargetOption>atSteadyState</steadyStateTargetOption>
            <targetExtractionOption>definitionIfNoIndividualTarget
                                                </targetExtractionOption>
            <formulationAndRouteSelectionOption>lastFormulationAndRoute
                                        </formulationAndRouteSelectionOption>
        </options>
    </adjustmentTraits>
</request>
```

The values are defined as follows:

| Element | Value inside |
|---|---|
| *requestId* | Constant. For example, "adjustment_" + < associated drug ID>. |
| *drugId* | Extracted from TuberXpert request element. |
| *drugModelId* | Retrieved by the TuberXpert drug model selection. |
| *parametersType* | From decisions presented previously. |
| *compartmentOption* | Always *allActiveMoieties.* |
| *retrieveStatistics* | Always *true.* To be used for the final report. |
| *retrieveParameters* | Always *true.* To be used for the final report. |
| *retrieveCovariates* | Always *true.* To be used for the final report. |
| *nbPointsPerHour* | Always *20*. |
| *start* | From decisions presented previously. |
| *end* | From decisions presented previously. |
| *adjustmentDate* | From decisions presented previously. |
| *bestCandidatesOption* | Always *bestDosagePerInterval.* |
| *loadingOption* | By default, follow drug model recommendation or retrieved from the TuberXpert request element. |
| *restPeriodOption* | By default, follow drug model recommendation or retrieved from the TuberXpert request element. |
| *steadyStateTargetOption* | From decisions presented previously. |
| *targetExtractionOption* | By default, *definitionIfNoIndividualTarget* or retrieved from the TuberXpert request element. |
| *formulationAndRouteSelectionOption* | By default, *lastFormulationAndRoute* or retrieved from the TuberXpert request element. |

## 4.5 Report

This chapter discusses the forms that the output will take. It is expected to be in the form of an XML document, an HTML page or a PDF document. As a first approach to understanding what information should be returned in the XML file, I produced a first draft of the HTML report page on Figma. From that point, it is possible to emphasis what information is necessary and deduce what will be inserted in the XML document.

Let's assume that the root element of the XML report is "tuberxpertResult". All the following XML structures will be inserted as children.

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| <tuberxpertResult> | | 1:1 | The data in the XML report. |

### 4.5.1 Header

This first part contains the date of computation as well as general facts about the drug concerned, such as its identifier, the last dose administered and the drug model selected for this treatment adjustment.

**HTML representation:**
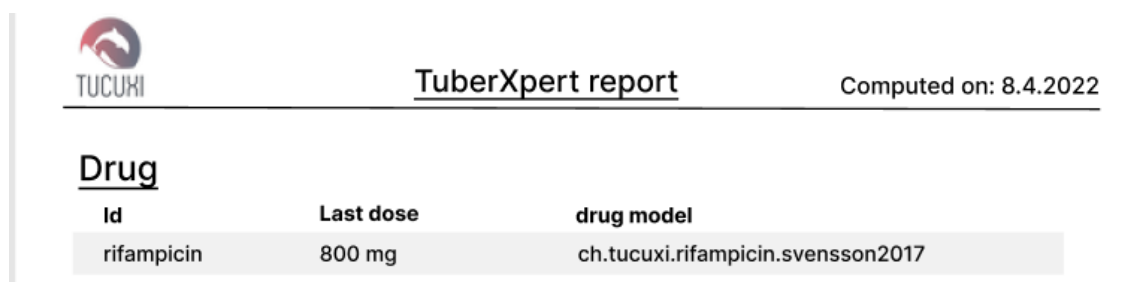


*Figure 11 - Report header*

**XML organization:**

```
<computationTime>2022-08-04T07:00:00</computationTime>
<language>en</language>
<drug>
    <drugId>rifampicin</drugId>
    <lastDose>
        <value>800</value>
        <unit>mg</unit>
    </lastDose>
    <drugModelId>ch.tucuxi.rifampicin.svensson2017</drugModelId>
</drug>
```

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| <computationTime> | date | 1:1 | The "present time" used. |
| <language> | string | 1:1 | The language of the translated elements. |
| <drug> | | 1:1 | Minimum drug information. |
| _<drugId> | string | 1:1 | The drug identifier. |
| _<lastDose> | | 1:1 | Last dose information. |
| __<value> | double | 0:1 | The value of the last dose. |

| __\<unit\> | string | 0:1 | The unit of the last dose. |
|---|---|---|---|
| _\<drugModelId\> | string | 1:1 | The selected drug model identifier. |

> *The \<language\> element is an enumeration. The values are "en" or "fr".*

The computationTime has the same value as the date element from the TuberXpert query.

Even if it is not present in the HTML version of the report. The desired output language will be included in the XML version with the language element. Unlike the HTML/PDF versions which are not likely to be reprocessed again, the XML version will probably be reprocessed again by another program. It is therefore a good idea to add any information needed for further processing. Without the language, we would not be able to know the language of the sentences and we would not be able to effectively add certain changes.

Another consequence of the fact that the XML version can be reprocessed is that not all standardized values will be translated. One can easily think of formulations, administration routes or target types.

If we turn the problem the other way around, the only things we want to translate are the warning messages because they are the only non-standardized data we have, the covariate names and the covariate descriptions because we do not want to incorporate all available translations.

## 4.5.2 Admin

It contains all the administrative data of the mandator, the patient and the clinical information. In fact, this part displays every administrative data found in the admin element of the input.

**HTML representation:**



*Figure 12 - Report administrative information*

**XML organization:**

The XML output should stick to the structure of the admin element from the TuberXpert query.

```xml
<admin>
    <mandator>
        <person>[…]</person>
        <institute>[…]</institute>
    </mandator>
    <patient>
        <person>[…]</person>
        <institute>[…]</institute>
    </patient>
    <clinicalDatas>[…]</clinicalDatas>
</admin>
```

The detailed description of this element is available [here](#).

### 4.5.3 Covariates

This section lists all the covariates that are needed for the adjustment computation. It indicates the value and the unit that will be used and the source of the covariate whether it is from the patient or from the drug model.

If the covariate is from the patient, it includes the measurement date. Also, if it does not respect the drug model validation, a warning is displayed.

**HTML representation:**



*Figure 13 - Report covariates information*

Here, there is an example of a default "fat-free mass" covariate. There is an alternative representation with a measurement date and a warning message.

**XML organization:**

```xml
<covariates>
    <covariate>
        <covariateId>ffm</covariateId>
        <name>Fat-Free Mass</name>
        <value>-10</value>
        <unit>kg</unit>
        <dataType>int</dataType>
        <desc>The fat-free mass of the patient</desc>
        <source>patient</source>
        <date>2022-07-11T13:45:30</ date >
        <warning level='normal'>The fat-free mass shall be positive.</warning>
    </covariate>
    […]
</covariates>
```

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| <covariates> | | 1:1 | Covariates specification. |
| _<covariate> | | 1: ∞ | Description of a covariate. |
| __<covariateId> | | 1:1 | The unique identifier of the covariate. |
| __<date> | date | 0:1 | If from the patient, the date of measure. |
| __<name> | string | 1:1 | The translated name of the covariate. |
| __<value> | string | 1:1 | The value of the covariate. |
| __<unit> | string | 1:1 | The unit of the covariate. |
| __<dataType> | string | 1:1 | The covariate value data type. |
| __<desc> | string | 1:1 | The translated description of the covariate. |
| __<source> | string | 1:1 | The source of the covariate. |
| __<warning> | string | 0:1 | The translated warning message if the validation is not respected. |

The <dataType> field is an enumeration. The values are "int", "double", "bool" or "date".

The <source> field is an enumeration. The values are "patient" or "default".

The <warning> element always has a "level" attribute with the value "normal".

## 4.5.4  Treatment

This section lists the doses from the patient's dosage history. It shows each dose within a dosage time range. It displays a warning for a dose if the value recommended by the drug model is reached.

**HTML representation:**



*Figure 14 - Report treatment information*

The challenge is to translate the treatment of the query into something visual.

The base dosage element of a dosage time range element can be a dosage loop or a dosage at steady state. If this is the case, the "Type" indication is displayed under the time range dates:

- <u>Loop</u>: continually
- <u>At steady state</u>: at steady state with last dose on…

For the doses, the main idea is to display an indication of the posology near each dose according to their type:

- <u>Lasting</u>: every + <interval>
- <u>Daily</u>: daily at + <time>
- <u>Weekly</u>: every <day> + at + <time>
- <u>Repeat</u>: <number of repetitions> time(s)

Since dosages can be nested, each achieved dosage will be added to the posology of the final dose. For example, if a daily dose is nested within a dosage repeat, the posology may be:

- 100g (oral) daily at 8h15, 4 time(s)

**XML organization:**

The output will return the treatment node as it entered with a small difference. Each suspicious dose element in a lasting/daily/weekly dosage node will receive an optional warning element with an error value.

For example, with a lastingDosage, the following situation could be possible:

```xml
<treatment>
    […]
    <lastingDosage>
        […]
        <dose>
            <value>1000</value>
            <unit>mg</unit>
            <infusionTimeInMinutes>60</infusionTimeInMinutes>
            <warning level='normal'>Maximum recommended dosage […]</warning>
        </dose>
        […]
    </lastingDosage>
</treatment>
```

> As already mentioned, the treatment is a complex element. This element follows the same structure as the element defined by Tucuxi. If you are interested in more details, see the file "tiersdoc/Tucuxi_CLI_Usability_Specification.pdf". The only element that changes is the dose element.

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| <dose> | | 1:1 | Description of a dose. |
| _<value> | decimal | 1:1 | The value of the dose. |
| _<unit> | string | 1:1 | The unit of the dose. |
| _<infusionTimeInMinutes> | decimal | 1:1 | The infusion time (min) of the dose. |
| _<warning> | string | 0:1 | Warning that describes the problem of the dose value. |

> The <warning> element always has a "level" attribute with the value "normal".

## 4.5.5  Samples

This section lists the patient's samples. It shows the date of the sample, its measure and the percentile to which it belongs. It displays a warning for a sample if it reaches some given threshold:

- Red warning if the percentile is below 5 or above 95
- Yellow warning if the percentile is below 10 or above 90

**HTML representation:**



*Figure 15 - Report samples information*

*For now, the Tucuxi computation core works with a single analyte. The day when the multiple analytes feature will be on rail, we must consider sorting the samples by analytes.*

**XML organization:**

The output will return the samples node as it entered with two differences. Each suspicious concentration will receive an optional warning element with an error value and each concentration will receive a new percentile element.

For example, the following situation could be possible:

```
<samples>
    <sample>
        <sampleId>2</sampleId>
        <sampleDate>2018-07-08T08:00:00</sampleDate>
        <concentrations>
            <concentration>
                <analyteId>rifampicin</analyteId>
                <value>700</value>
                <unit>mg/l</unit>
                <percentile>100</percentile>
                <warning level='critical'>99% of the population is […]</warning>
            <concentration>
            […]
        </concentrations>
    </sample>
    […]
</ samples >
```

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| \<samples\> | | 1:1 | The samples of the drug. |
| _\<sample\> | | 0: ∞ | Description of a sample. |
| __\<sampleId\> | int | 1:1 | The unique identifier of the sample. |
| __\<sampleDate\> | date | 1:1 | The date of the sample. |
| __\<concentrations\> | | 1:1 | Concentrations. |
| ___\<concentration\> | | 1:∞ | Concentration specification. |
| ____\<analyteId\> | string | 1:1 | The unique identifier of the analyte. |
| ____\<percentile\> | int | 1:1 | The group number over the 99 percentiles. |
| ____\<value\> | decimal | 1:1 | The value of the concentration. |
| ____\<unit\> | string | 1:1 | The unit of the value. |
| ____\<warning\> | string | 0:1 | Warning of the sample percentile. |

> *The \<percentile\> field contains a value from 1 to 100.*

> *The \<warning\> element always has a "level" attribute:*
> - *"critical": If the sample is below the 5th or above the 95th percentile.*
> - *"normal": Otherwise.*

## 4.5.6  Best adjustments and suggested adjustment

This section is divided into two main parts:

- The best adjustment per interval.
- The suggested adjustment.

Each of these parts begins the same way:

- A brief introduction.
- A graph that displays the adjustment predictions.
- The treatments that match to the displayed predictions.

Each adjustment receives a score assigned by Tucuxi computation core. Per adjustment, the targets used get a score according to the following formula:

$$Score = 1 - \frac{(\log(C_{predicted}) - \log(C_{target}))^2}{(0.5\log(C_{up}) - 0.5\log(C_{lo}))^2}$$

*"The idea is that this score reflects the squared relative departure of the predicted concentration from the target value at the corresponding time, expressed in the scale of the relative radius of the therapeutic interval."*

In other words, if the prediction tends towards the target, the score will tend towards 1, otherwise towards 0. The ease with which the score of 1 is achieved is mitigated by the extent of acceptance of the target. The higher the score, the better.
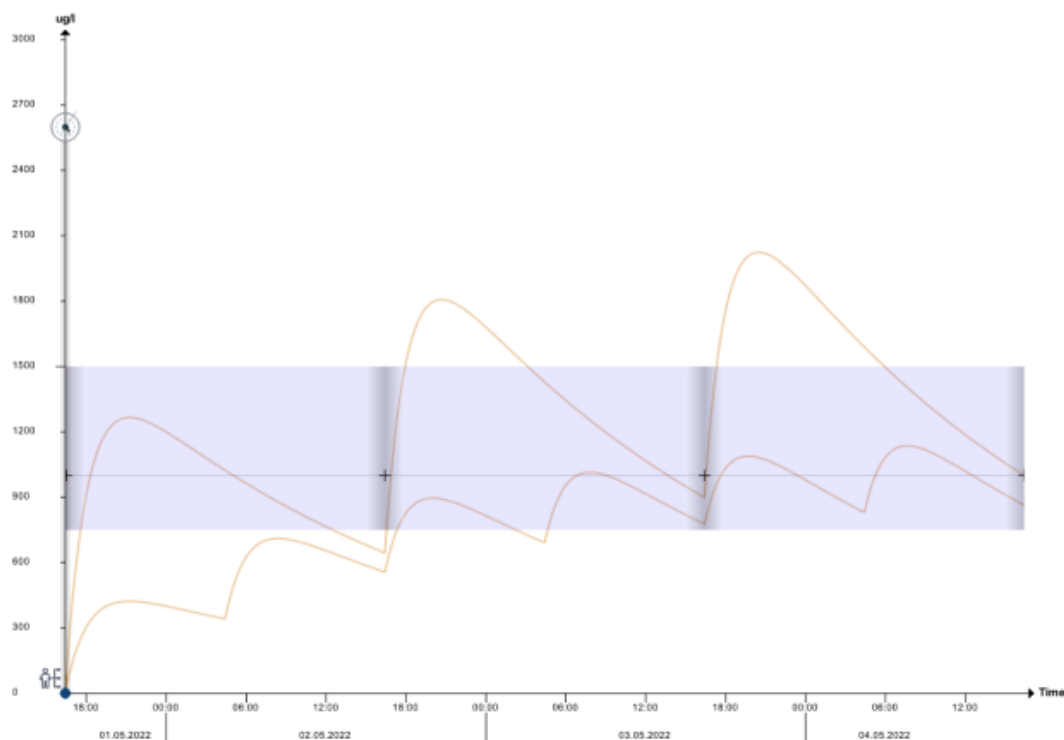
Finally, when each target has a score, the adjustment score is obtained from the average of all target scores.

**HTML representation of the best adjustment per interval:**



*Figure 16 - Report adjustment per interval information*

We can see the score at the top of the adjusted treatments. The way the adjusted treatments are displayed is the same as the patient treatment is displayed.

If the Tucuxi computation core finds only one adjustment, this part of the HTML will be dropped because it will strictly display the same graph and the same adjusted treatment as the next part.

**HTML representation of the suggested adjustment:**



## Adjustment suggested

Considering the above information, TuberXpert recommends the following adjustment.
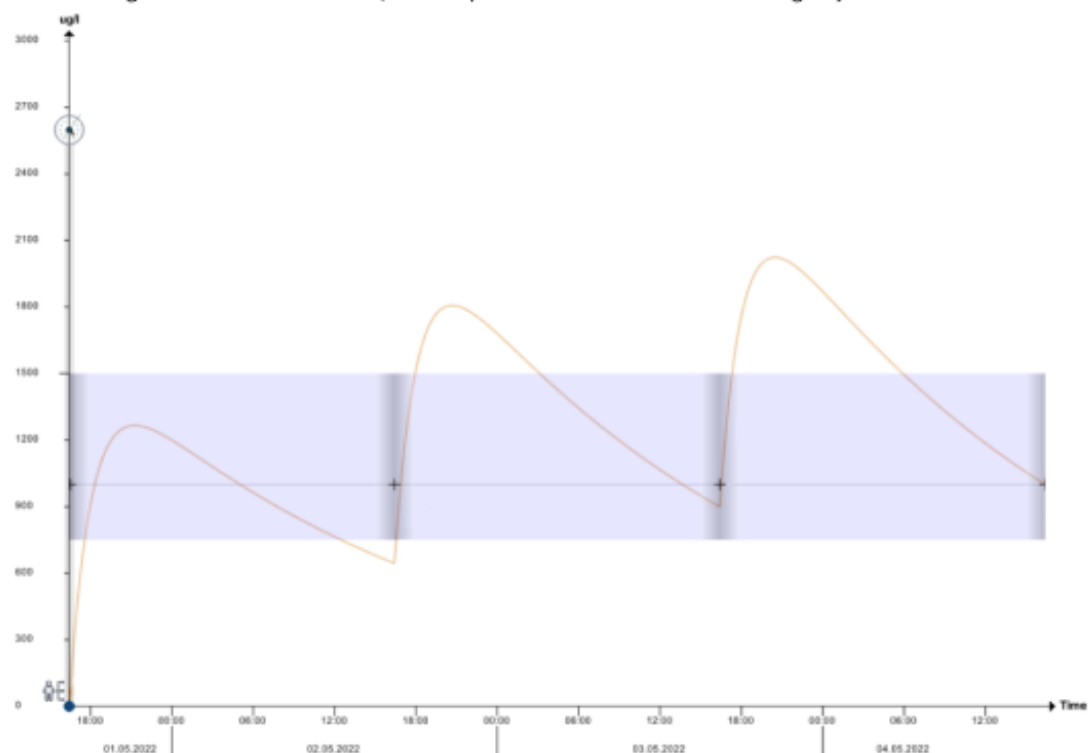
### Displayed adjustment

**Score:** 0.95 / 1

- **From:** 1.5.2022 16h25      **To:** 6.5.2022 16h25
  **Posology:** 800 mg (oral), 1 time(s)

- **From:** 1.5.2022 16h25      **To:** 6.5.2022 16h25
  **Posology:** 600 mg (oral) every 24h, 7 time(s)

With the best suggestion, the targets are expected to change as follows.

| Type | Value (unit) | Score |
|------|--------------|-------|
| Residual | 1103.12 (ug/l) | 0.95 / 1 |
| Inefficacy: 0 / **Min: 750** / **Best: 1000** / **Max: 1500** / Toxicity: 10000 | | |

*Figure 17 - Report suggested adjustment information*

This second part highlights the adjustment with the highest score already displayed in the best adjustment per interval part.

Additionally, it displays the achievement of the targets. For each target, it lists its type, its predicted value, its score and the inefficacy and toxicity limits.

**XML organization:**

The XML content that contains information for the previous two parts should be approximately the same as the "dataAdjustment" element of the Tucuxi computation response. For example, the following XML structure is possible:

```
<dataAdjustment>
    <analyteIds>
        <analyteId>rifampicin</analyteId>
    </analyteIds>
    <adjustments>
        <adjustment>
            <score>0.985</score>
            <targetEvaluations>[See below]</targetEvaluations>
            <dosageHistory>[See below]</dosageHistory>
            <cycleDatas>[See below]</cycleDatas>
        </adjustment>
        […]
    </adjustments>
</dataAdjustment>
```

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| <dataAdjustment> | | 1:1 | The data of the adjustments found. |
| _<analyteIds> | | 1:1 | Analyte id container. |
| __<analyteId> | string | 0:∞ | The unique identifier of an adjusted analyte. |
| _<adjustments> | | 1:1 | Adjustments container. |
| __<adjustment> | | 1:∞ | Adjustment description. |
| ___<score> | double | 1:1 | The score of the adjustment. |
| ___<targetEvaluations> | TargetEvaluations | 1:1 | Target evaluations. |
| ___<dosageHistory> | DosageHistory | 1:1 | Dosage history. |
| ___<cycleDatas> | CycleDatas | 1:1 | Cycle data list. |

The "targetEvaluations" element contains a list of "targetEvaluation" elements that represent the targets that the adjustment is trying to achieve.

```
<targetEvaluations>
    <targetEvaluation>
        <targetType>residual</targetType>
        <unit >ug/l</unit>
        <value>1103.122367</value>
        <score>0.919806</score>
        <min>750</min>
        <best>1000</best>
        <max>1500</max>
        <inefficacyAlarm>0</inefficacyAlarm>
        <toxicityAlarm>10000</toxicityAlarm>
    </targetEvaluation>
    […]
</targetEvaluations>
```

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| <targetEvaluation> | | 1:∞ | Target evaluation specification. |
| _<targetType> | string | 1:1 | The type of the target. |
| _<analyteId> | string | 1:1 | The unique identifier of an analyte. |
| _<unit> | string | 1:1 | The unit of the value. |
| _<value> | double | 1:1 | The expected value of the target. |
| _<score> | double | 1:1 | The score of the target. |
| _<min> | double | 1:1 | The minimum value to reach. |
| _<best> | double | 1:1 | The ideal value. |
| _<max> | double | 1:1 | The maximum value to not reach. |
| _<inefficacyAlarm> | double | 1:1 | The inefficacy limit. |
| _<toxicityAlarm> | double | 1:1 | The toxicity limit. |

*The target type is an enumeration. The values are "peak", "residual", "mean" or "auc", "aucOverMic", "timeOverMic", "aucDividedByMic" or "peakDividedByMic".*

The dosage history element contains the treatment to be followed for the adjustment. Its structure is the same as Tucuxi query dosage history.

*For a complete Tucuxi dosage history specification, see "tiersdoc/Tucuxi_CLI_Usability_Specification.pdf".*

```
<dosageHistory>
    <dosageTimeRange>
        <start>2018-01-12T07:00:00</start>
        <end>2018-03-15T12:59:00</end>
        <dosage>[…]</dosage>
    </dosageTimeRange>
    […]
</dosageHistory>
```

The predictions for a given adjustment are contained in cycleData elements in the cycleDatas element.

A cycleData represents the predictions between two intakes. It contains a start and end date, the unit of the values, a times element that contains a list of times, in hours, starting from zero and a values element that contains a list of predicted concentration values. The number of times matches the number of values.

```
<cycleDatas>
    <cycleData>
        <start>2018-01-12T07:00:00</start>
        <end>2018-01-12T19:00:00</end>
        <unit>ug/l</unit>
        <times>[…]</times>
        <values>[…]</values>
    </cycleData>
    […]
</cycleDatas>
```

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| <cycleData> | | 0:∞ | Cycle data specification. |
| _<start> | date | 1:1 | The start date of the cycle data. |
| _<end> | date | 1:1 | The end date of the cycle data. |
| _<unit> | string | 1:1 | The unit of the concentration values. |
| _<times> | string | 1:1 | A list of times. |
| _<values> | string | 1:1 | A list of concentration values. |

> *The <times> field contains a comma-separated list of times, in hours, starting from 0. To get the real time of a concentration, this time should be added to start.*

> *The <values> field contains a comma-separated list of concentration values. The number of values matches the number of times.*

## 4.5.7 Computation facts

This is the last section of the report. It contains general facts about the computation, such as the pharmacokinetic parameters, some steady state predictions and the covariates used by the computation core. This second list of covariates is useful because it allows to double check the covariates. In addition, some covariates can be calculated on running time based on the patient's covariates. Thus, it is possible to see if there are any calculated covariates.

**HTML representation:**

### Pharmacokinetic parameters

| | Typical patient | A priori | A posteriori |
|---|---|---|---|
| ● CL | 15.106 | 15.202 | 15.202 |
| ● F | 1 | 1 | 1 |

### Predictions

| | |
|---|---|
| **Extrapolated steady state AUC24** | 36863.6 ug*h/l |
| **Steady state peak** | 2023.58 ug/l |
| **Steady state residual** | 999.701 ug/l |

### Covariates used for computation

| Covariate | value |
|---|---|
| ● Age | 50 |
| ● Total body weight | 40 |
| ● GIST | No |
| ● Sex | Man |

*Figure 18 - Report computation facts information*

**XML organization:**

```
<parameters>
    <typical>
        <parameter>
            <id>Ka</id>
            <value>0.609</value>
        </parameter>
        […]
    </typical>
    <apriori>
        <parameter>[…]</parameter>
        […]
    </apriori>
    <aposteriori>
        <parameter>[…]</parameter>
        […]
    </aposteriori>
</parameters>
<statistics>
    <auc24>36863.6</auc24>
    <peak>2023.58</peak>
    <residual>999.701</residual>
</statistics>
<computationCovariates>
    <computationCovariate>
        <id>bodyweight</id>
        <value>40.000000</value>
    </computationCovariate >
    […]
</computationCovariates>
```

The parameters element contains a listing of each pharmacokinetic parameter for each computation type: typical, apriori and aposteriori. Depending on the parameters type of the request, additional adjustment requests are required to obtain all parameters. For example, if the current adjustment is "A posteriori", additional adjustment requests are made to get the "Typical patient" and "A priori" parameters.

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| <parameters> | | 1:1 | The pharmacokinetic parameters. |
| _<typical> | | 1:1 | The parameters of the typical patient. |
| __<parameter> | | 1:∞ | A pharmacokinetic parameter. |
| ___<id> | string | 1:1 | The identifier of the parameter. |
| ___<value> | string | 1:1 | The value of the parameter. |
| _<apriori> | | 1:1 | The parameters "A priori". |
| __<parameter> | | 1:∞ | A pharmacokinetic parameter. |
| ___<id> | string | 1:1 | The identifier of the parameter. |
| ___<value> | string | 1:1 | The value of the parameter. |

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| _<aposteriori> | | 0:1 | The parameters "A posteriori". |
| __<parameter> | | 1:∞ | A pharmacokinetic parameter. |
| ___<id> | string | 1:1 | The identifier of the parameter. |
| ___<value> | string | 1:1 | The value of the parameter. |

The statistics element contains the predictions at steady state. The steady state can be approximated using the formula:

$$Time\ of\ steady\ state = adjustment\ time + 2 * drug\ half Life * multiplier$$

The half-life and the multiplier are temporal considerations located in the drug model. The half-life is the time it takes for the amount of a drug's active substance to reduce by half.

These statistics are computed in each cycle data returned by the Tucuxi computation core. It is thus necessary to make another adjustment request, but in a light way, to obtain them.

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| <statistics> | | 1:1 | The statistics at steady state. |
| _<auc24> | double | 1:1 | Area under the curve on 24h. |
| _<peak> | double | 1:1 | Peak concentration. |
| _<residual> | double | 1:1 | Residual concentration. |

The computationCovariates element lists all the covariates used during the computation, represented by a computationCovariate element which contains a value and the id of covariate. These values can be found in any adjustment data returned by Tucuxi computation core.

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| <computationCovariates> | | 1:1 | The covariates during computation. |
| _<computationCovariate> | | 1:∞ | A computation covariate. |
| __<id> | string | 1:1 | The identifier of the covariate. |
| __<value> | double | 1:1 | The value of the covariate. |

> *The PDF/HTML version converts the double value of the computation covariate into something more natural when possible.*
> *For example:*
> - *If the covariate is a sex, it becomes "Man", "Woman" or "Undefined".*
> - *If the covariate type is a bool, it becomes "Yes" or "No".*

# 5 <u>Implementation</u>

With a walk-through of an execution, this chapter presents the classes in TuberXpert and their utility. Sometimes there are class diagrams to better visualize the interactions. However, these diagrams are not meant to fully present the content of the classes. They will only contain what is necessary to understand the purpose of the class.

> *For a complete description of the classes, there is an automatically generated code documentation available in "publi/TuberXpert_code_documentation.html".*

The TuberXpert implementation is available in "dev/tucuxi-tuberxpert". Inside this directory is the file "tucuxi-tuberxpert.pro" which allows to open the project in Qt Creator.

The organization of the project is as follows:

- It includes the source code of Tucuxi CLI.
- The source code of TuberXpert is accessible by the "tuberxpert" inclusion.
- The master program is in a single file "tuberxpert.cpp." It contains the main function that launches the execution.
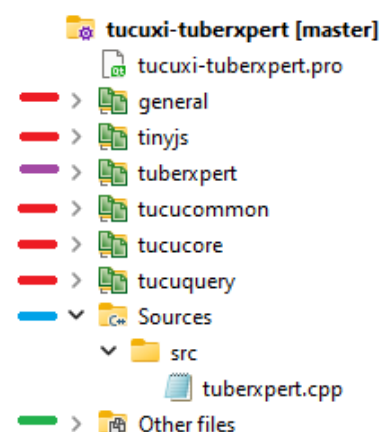- The "Other files" folder contains some XML translations files and some XML validation files.



*Figure 19 - TuberXpert project in Qt Creator*

At the beginning of each chapter, there is a table that summarizes the important files. To lighten the annotations, we assume that all paths are relative to "dev/tucuxi-tuberxpert".

## 5.1 Run TuberXpert

**Concerned files**

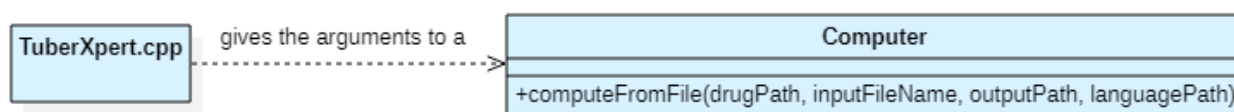| /src/tuberxpert.cpp |
| --- |
| /src/tuberxpert/computer.h |



*Figure 20 - Class diagram TuberXpert program starting*

TuberXpert works as CLI that expects the following arguments:

> **-d \<path/to/drug/models\>** to indicate where the drug model files are. There is a basic collection in "dev/tucuxi-drugs/drugfiles".
>
> **-i \</path/to/xml/query\>** to indicate the location of the TuberXpert query to process. There are a few examples in "dev/tucuxi-tuberxpert/xml/query".
>
> **-o \</path/to/directory\>** to indicate where the result reports should be printed.
>
> **-l \</path/to/translations/files\>** to indicate where the translations files are. Actually, TuberXpert supports English and French in "dev/tucuxi-tuberxpert/language".

The main file "tuberxpert.cpp" analyses the arguments and transmits them to the TuberXpert computer which launches the TuberXpert flow.

> *The class "Computer" also provides a "computeFromString" method that accepts the contents of a TuberXpert query as a string instead of the file name. Therefore, it could be easily implemented on web server.*

# 5.2 Query import

**Concerned files**

/src/tuberxpert/query/admindata.h

/src/tuberxpert/query/xpertquerydata.h

/src/tuberxpert/query/xpertqueryimport.h

/src/tuberxpert/query/xpertrequestdata.h

The first task of the TuberXpert computer is to import the TuberXpert query. For a description of the query file, see here. But first, let's look at all the classes involved during the import.

> *The structure of the query can be validated with the file "tuberxpert_computing_query.xsd" in "dev/tucuci-tuberxpert/xml/query".*

## 5.2.1 Administrative data

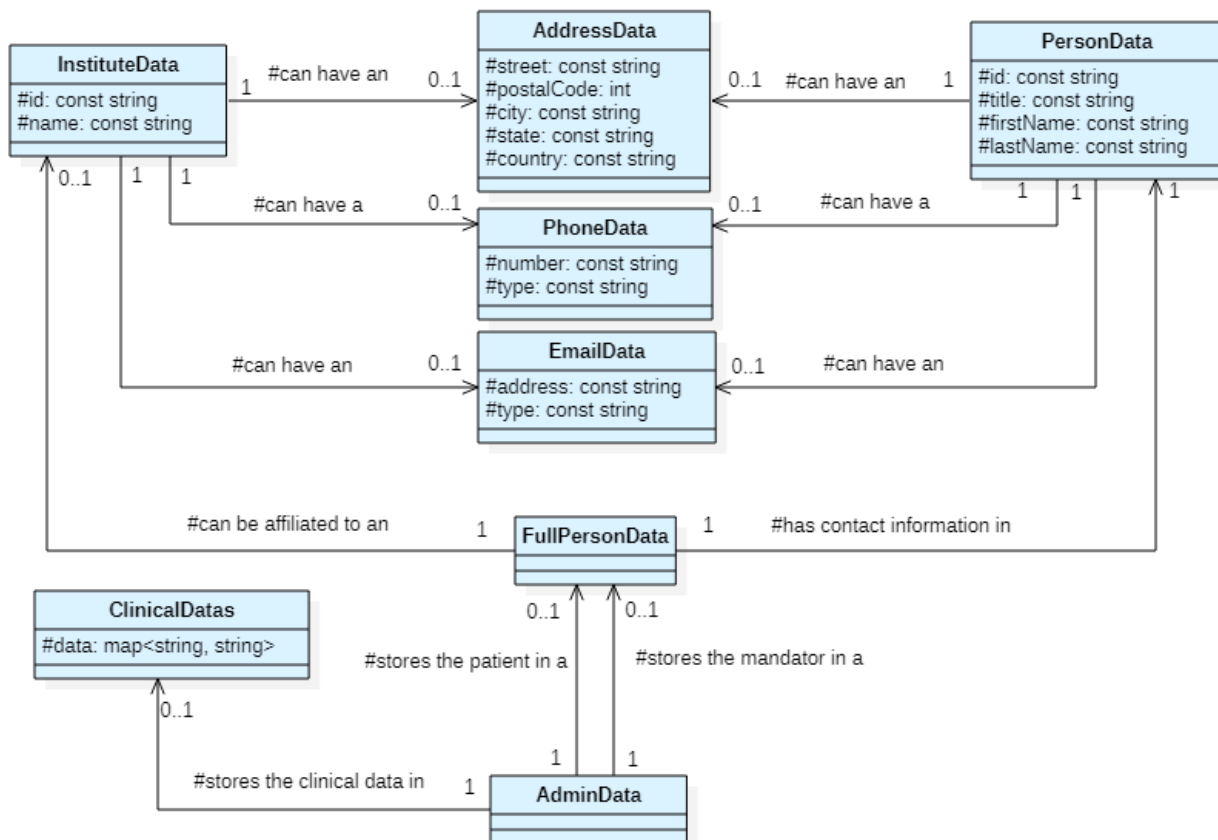The *AdminData* class provides all the information of the admin element of the TuberXpert query.



*Figure 21 - Class diagram of administrative data*

The class modeling follows the same structure as the query file. The admin element is represented by the *AdminData* class. The *ClinicalDatas* class contains a map for each *clinicalData* found in the *clinicalDatas* element. The attribute *key* is used as the input key of the map and the value of *clinicalData* as the input value of the map. The *FullPersonData* class is used for the mandator and the patient elements. The *PersonData* and *InstituteData* classes contain data from the person and institute elements. The *AddressData*, the *PhoneData* and *EmailData* classes encapsulate the information contained in the address, phone and email elements.

## 5.2.2 Request data

The *XpertRequestData* class provides all the information of an *xpertRequest* element of the TuberXpert query.
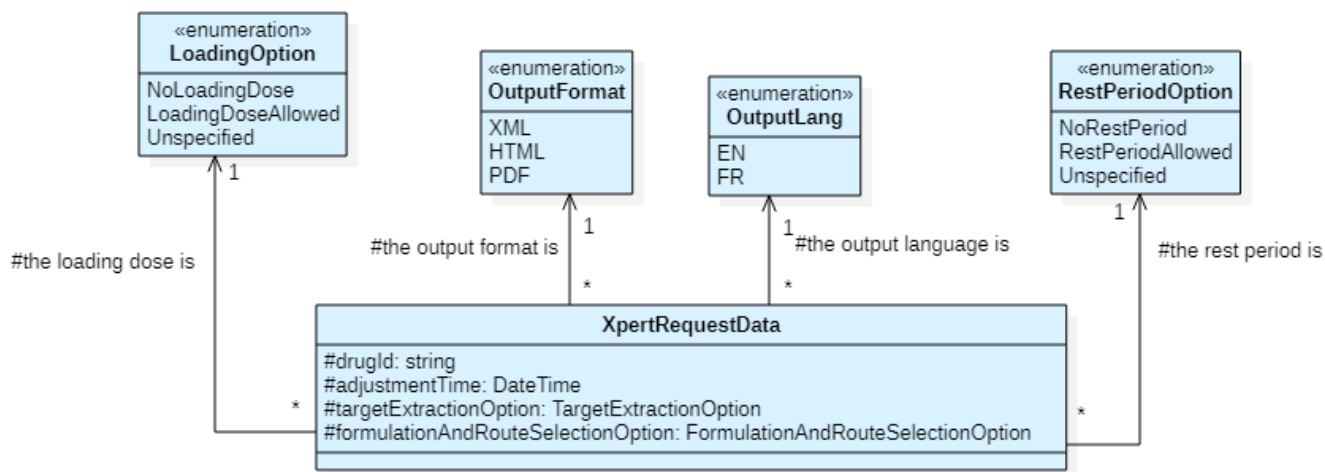


*Figure 22 - Class diagram of the request data*

The *xpertRequest* element is represented by the *XpertRequestData* class. The output format and language are represented by the enumerations *OutputFormat* and *OutputLang*. The *loadingOption* and *restPeriodOption* values are translated into the *LoadingOption* and the *RestPeriodOption* enumerations. The value "Unspecified" is used when the user does not explicitly allow or disallow these options. This means that the recommendations of the drug model should be used. Finally, the *targetExtractionOption* and the *formulationAndRouteSelectionOption* use the enumerations implemented in Tucuxi computation core. Thus, the values are not displayed on this UML.

## 5.2.3 Query data

The *XpertQueryData* class is the C++ equivalent of the TuberXpert XML query.

Since the TuberXpert query is a specialization of the Tucuxi query, the TuberXpert query class inherits from the Tucuxi query class. Therefore, all the structure for the common elements is ready, we just need the *AdminData* and *XpertRequestData* class to store the additional elements
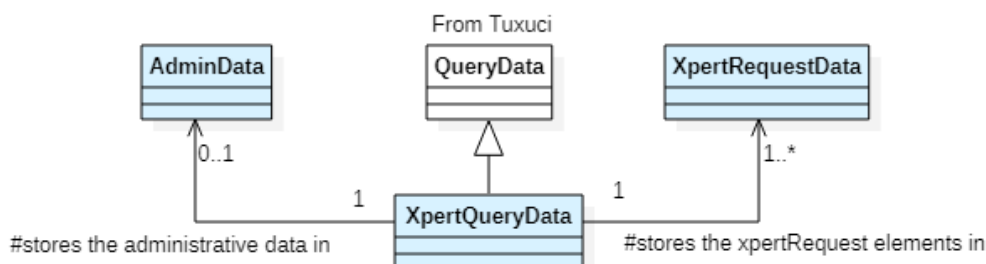


*Figure 23 - Class diagram of the query*

## 5.2.4 Query import

The *XpertQueryImport* class is responsible for importing the TuberXpert query. In the same way that the TuberXpert query class inherits from the Tucuxi query class, the importer inherits from the Tucuxi importer.

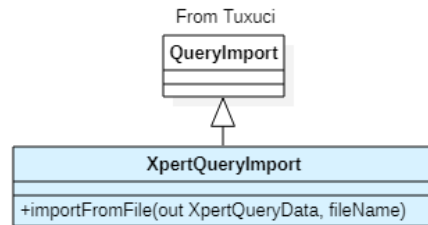In this way, we can use the existing implementation for the common elements.



*Figure 24 - Class diagram of the query importer*

> *In response to the "computeFromString" method of the "Computer" class, the class "XpertQueryImport" also provides an "importFromString" method that accepts the contents of a TuberXpert query as a string instead of a file name. Therefore, it could be easily implemented on web server.*
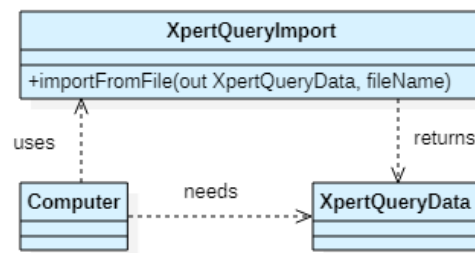
## 5.2.5 Import overview



*Figure 25 - Class diagram of simplified query import*

The TuberXpert computer must import the TuberXpert query as an *XpertQueryData* which is the C++ equivalent of the TuberXpert query. To do this, the TuberXpert computer uses the *XpertQueryImport* class, specially designed for the import of TuberXpert query.

# 5.3 Create the results holders

**Concerned files**

/src/tuberxpert/result/abstractvalidationresult.h

/src/tuberxpert/result /covariatevalidationresult.h

/src/tuberxpert/result /dosevalidationresult.h

/src/tuberxpert/result /samplevalidationresult.h

/src/tuberxpert/result /xpertqueryresult.h

/src/tuberxpert/result /xpertrequestresult.h

/src/tuberxpert/query /xpertquerytocoreextractor.h

Before we begin any execution of the flow steps, we need a way to store the information needed to print the reports. Let's explore this implementation from the macro to the micro level.

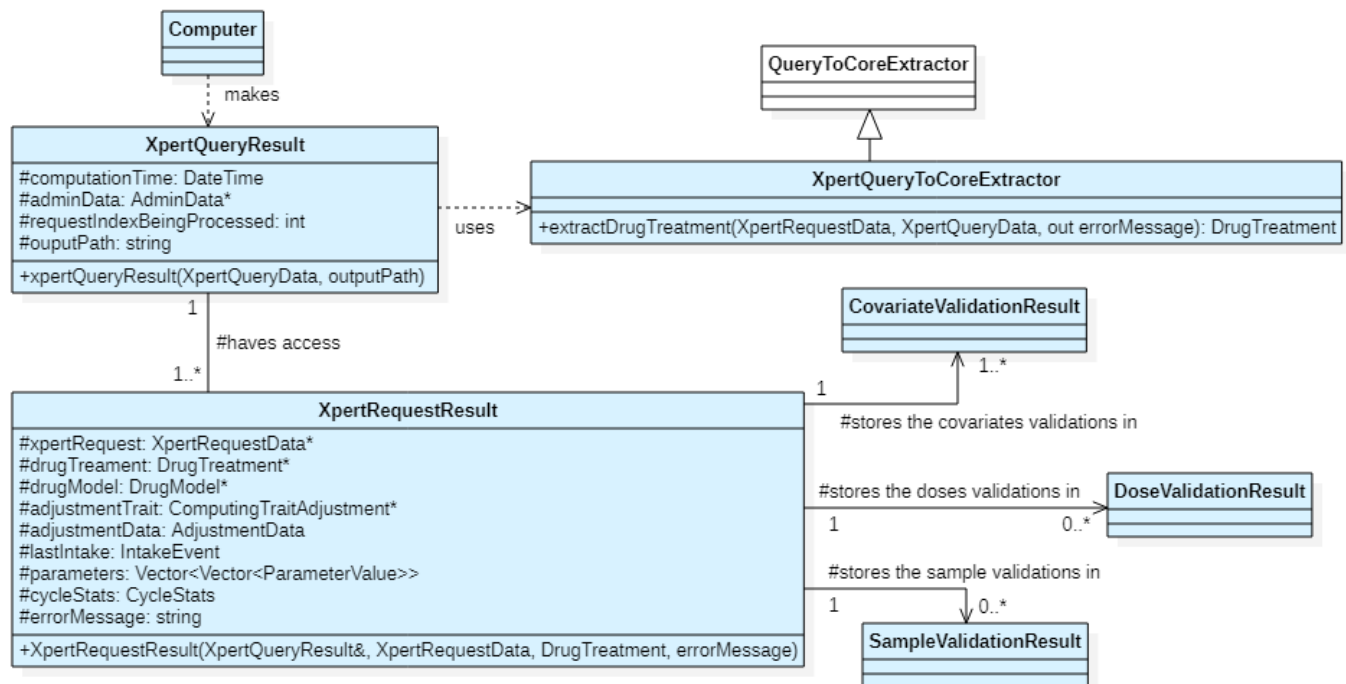## 5.3.1 Query result and request results



*Figure 26 - Class diagram of query result and request results*

When the TuberXpert computer has successfully imported the query in an *XpertQueryData*, it creates the objects that will be used during the whole process: from the drug model selection until the report generation.

***XpertQueryToCoreExtractor***

This class is in charge of creating a *DrugTreatment* from an *XpertQueryData* based on a drug identifier in an *XpertRequestData*. In fact, this extraction phase is necessary to have objects that can be processed by the Tucuxi calculation core. A drug treatment contains the patient's *DosageHistory*, *PatientCovariate* objects, *Target* objects and *Sample* objects.

It has inherited *QueryToCoreExtractor* to add some checks before extracting the treatment. For example, if it tries to extract a treatment from a drug that does not exist or exists multiple times in the *XpertQueryData*, there will be a specific error message that will be logged.

### XpertQueryResult

The idea behind this class is to store the common data that are needed by all the *XpertRequestResult* objects.

At creation, for each *xpertRequest*, the *XpertQueryResult* extracts the related *DrugTreatment* objects for processing, creates an XpertRequestResult and places the potential error returned by *XpertQueryToCoreExtractor*.

### XpertRequestResult

The *XpertRequestResult* class is the key class of TuberXpert. This class will go through each step of the flow, provide all the necessary information and gather their results to finally generate the adjustment report of an *xpertRequest*.

When it is created, it only contains the reference to the *XpertRequestResult* that created it and the drug treatment it is trying to adjust.

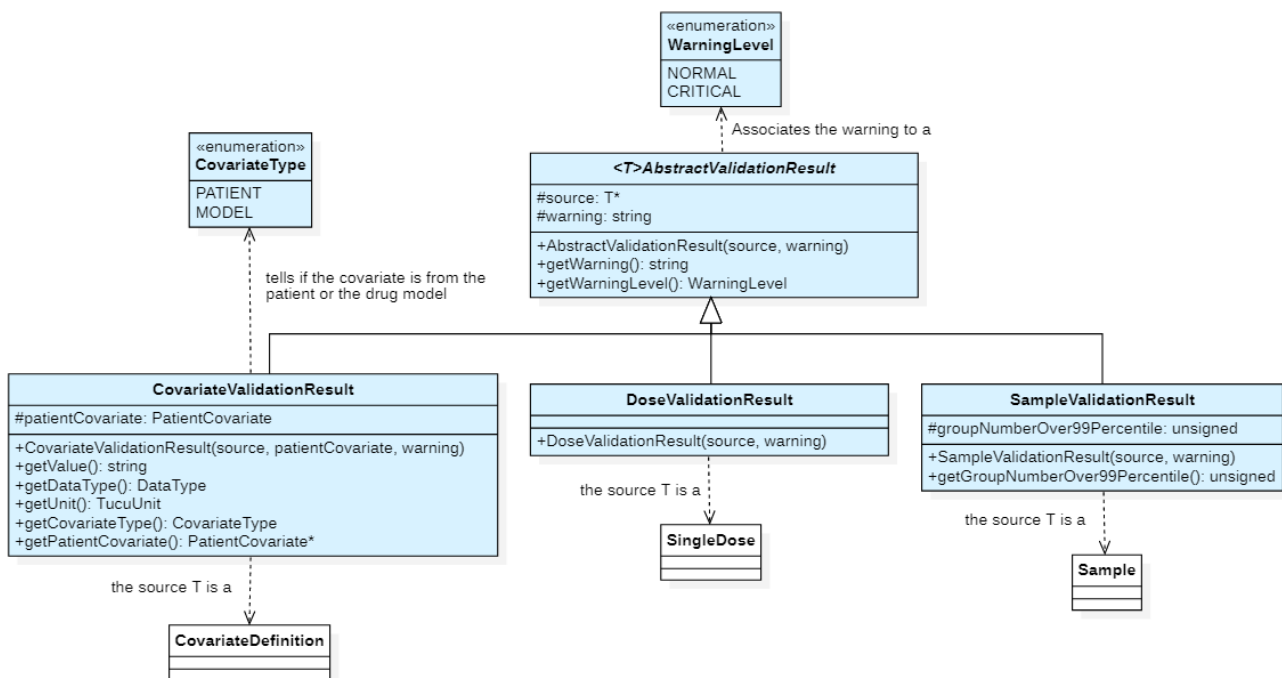## 5.3.2  Covariates, doses and samples validation results



*Figure 27 - Class diagram of the covariates, doses and samples validation result classes*

If we remember correctly, we need to provide a validation to the covariates, the doses and the samples. This is the purpose of these three classes. The pattern is to have objects that provides additional information to already existing objects.

> *A good question to ask is: Why are we using the composition instead of the inheritance ? For two reasons:*
>
> 1) *As we have seen previously in this section, there is a phase that extracts the "XpertQueryData" into "DrugTreatment" object that contains "PatientCovariate", "SingleDose" and "Sample" objects. We do not have control over this phase and it was not necessary to rewrite it for our needs.*
> 2) *For the "CovariateValidationResult" objects, we need to know which drug model we are using with the given treatment. This is not possible to know before extracting the "DrugTreatment" from the "XpertQueryData".*
>
> *So, with the composition we have more flexibility.*

*CovariateValidationResult*

One important thing to understand is that when a prediction is made, the Tucuxi computation core always needs at least one value for each associated drug model covariate definition. However, the source of the covariate makes no difference, whether it comes from the patient or from the default values of the drug model. So, we have to make the difference ourselves. Once we select our drug model, this class will allow us to make the difference while having the ability to attach warning messages if necessary.

There is going to be one object per covariate definition that is not defined in the patient covariates. In this case, the *PatientCovariate* is null pointer, there is no error message, the *getCovariateType* method will return "model" and the other methods will return the default values provided by the covariate definition.

There is going to be one object per patient covariate that corresponds to a drug model covariate definition. In this case, the *PatientCovariate* is not null pointer, there could be an error message if the covariate value does not meet the validation of the covariate definition. The *getCovariateType* method will return "patient" and the other methods will return the values provided by the patient covariate.

In any case, the warning level is always "normal".

This object allows to be very flexible with the covariate, whether or not there is a corresponding patient covariate. We can get the type of a covariate, its value and its warning message.

*DoseValidationResult*

This class simply allows to link a warning message to a dose.

In any case, the warning level is always "normal".

*SampleValidationResult*

For each patient blood sample, this class stores the position of the sample relative to the 100 groups formed by the 99 possible percentiles from 1 to 100. For example, if the sample is before the first percentile, its position will be 1. If its position is 100, the sample is above the 99 percentiles.

In addition, there is a warning message if the group is <= 10 or > 90. Depending on the position of the sample, the warning level is either "normal" or "critical":

- "critical": if the position is <= 5 or > 95.
- "normal": otherwise.

*Do you remember? This validation results class looks suspiciously like a class that provides all the information we need for the report. Take a look at the possible XML output of covariates, doses and samples.*

# 5.4 Select the correct flow steps

**Concerned files**

/src/tuberxpert/flow/abstract/abstractxpertflowstep.h

/src/tuberxpert/flow/abstract/abstractxpertflowstepprovider.h

/src/tuberxpert/flow/general/generalxpertflowstepprovider.h

At that time, the TuberXpert computer created the holders of the results that do not contain much:

- *XpertQueryResult*: the administrative data, the computation time, the output path and the index of the *XpertRequestResult* objects being processed.
- *XpertRequestResult*: The TuberXpert request for which this object contains results and the drug treatment for the TuberXpert request.

Now, thanks to an abstract factory system, we are going to get a provider of the flow steps that we are going to execute for a given *XpertRequestResult*.

> *Do you remember? TuberXpert is developed in a general way without targeting the behavior of a specific drug. In the future, with this abstract factory system, we can easily imagine having a custom flow step provider that offers a more specific implementation than the general system.*

## 5.4.1 Abstract flow step and abstract flow step provider
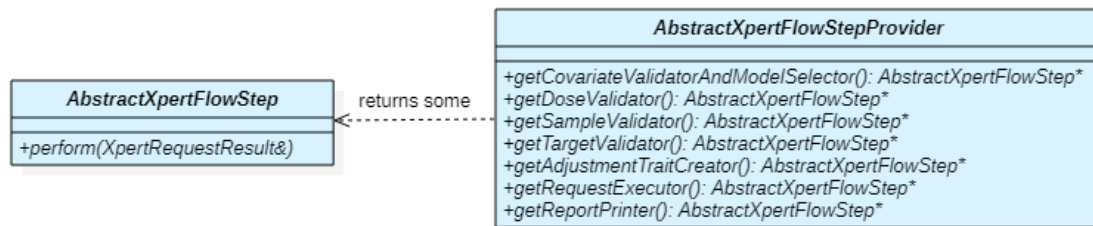


*Figure 28 - Class diagram of the abstract factory class*

### *AbstractXpertFlowStep*

The idea is that all the steps in the flow have the same interface to be easily replaced or extended. Thanks to the *XpertRequestResult* which has access to all the data to be validated and the results of the validation, this is possible. Therefore, a flow step to be performed needs to inherit from *AbstractXpertFlowStep*.

### *AbstractXpertFlowStepProvider*

This is the class we need to inherit from to be a step provider. As we can see, there is one getter method per flow step we plan to execute.

> *Do you remember? It seems that the "AbstractXpertFlowStepProvider" has the methods that return what we need to execute the steps that we have planned in the section Program execution flow.*
>
> *You got the idea, to fully process an "xpertRequest", we have to go through all these steps.*

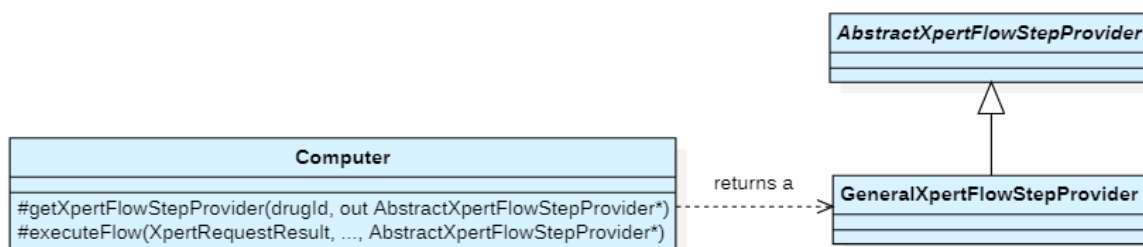## 5.4.2 Overview of the flow step provider selection



*Figure 29 - Class diagram of the TuberXpert computer retrieving the flow steps*

After creating the results holders, TuberXpert starts processing each *XpertRequestResult* with the *executeFlow* method, one after the other. Before, it needs to select the correct flow step provider that fit the most to the related drug. The *getXpertFlowStepProvider* method is in charge to return a concrete instance of an *AbstractXpertFlowStepProvider* that matches the given drug identifier.

> *In fact, the "getXpertFlowStepProvider" method returns a "GeneralXpertFlowStepProvider" in all cases. But in the future, if a drug needs a flow that differs a little bit, everything has been provided to make this possible.*

## 5.5 Process a request result

**Concerned files**

| |
|---|
| /src/tuberxpert/language/* |
| /src/tuberxpert/flow/general/* |
| /src/tuberxpert/exporter/* |

At the very beginning of the *executeFlow* method of the TuberXpert computer, before executing the flow steps (drug model selection with covariate validation until report generation), the first thing to do is to load the translations into a singleton that can be used anywhere and at any time. This is especially useful for translating dose and sample warnings and HTML/PDF report content. First, let's see what a translation file is.

### 5.5.1  Load a translations file

A translations file is an XML that contains a list of translation elements. Each element has an attribute that stores a key, and a value that is the actual translation.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<translations xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:noNamespaceSchemaLocation="translations_file.xsd">


        <translation key="A_key" >A translation</entry>
        […]


</translations>
```

| Tag name | Format | Occ. | Description |
|---|---|---|---|
| < translations > | | 1:1 | Root element. |
| _< translation > | string | 0: ∞ | A translation to be retrieved thanks to the key attribute. |

*Additionally, the content of the translations file could be validated with the file "dev/tucuxi-tuberxpert/language/translations_file.xsd".*
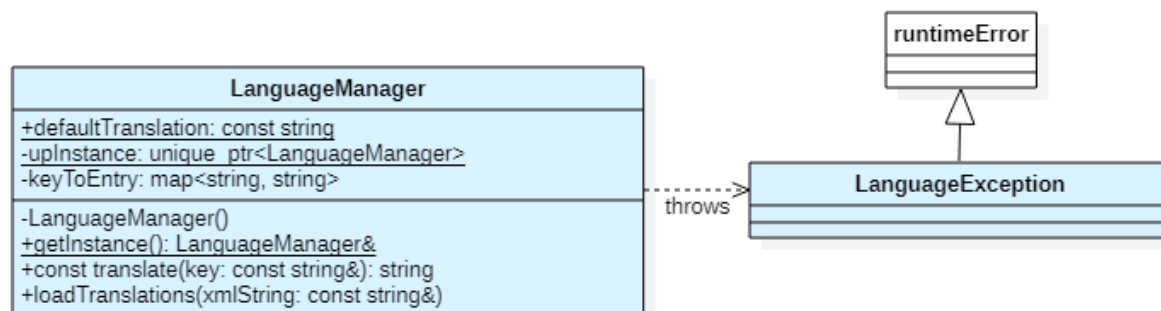
**Language manager**



*Figure 30 - Class diagram of the language manager*

The language manager is a singleton class that is responsible for maintaining the translations during the processing of an *xpertRequest* during the entire flow. The *loadTranslations* method allows to load a translations file as a string. The translate method is used to retrieve a translation for a given key. If the key does not exist, a default translation string is returned. If the language manager fails to load the translations, it throws a *LanguageException*.
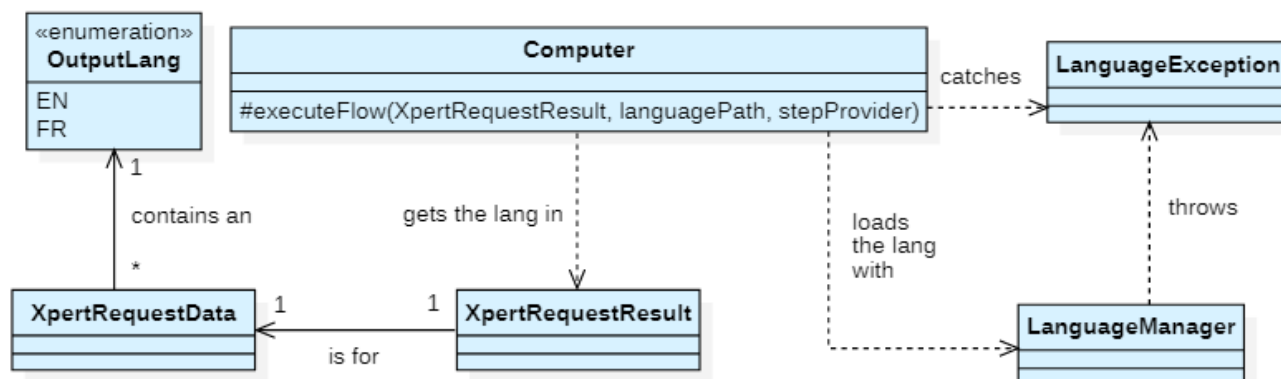
**Simplified overview**



*Figure 31 - Class diagram of the TuberXpert computer loading a language*

The TuberXpert computer gets the language to be loaded through the *XpertRequestResult* it is processing. It converts the output language into a string. Then it tries to read the corresponding translations file in the directory pointed by the *languagePath* argument and loads the contents of the resulting string with the *LanguageManager*. If it catches a *LanguageException*, the processing of the *XpertRequestResult* is aborted.

> *You may ask: How do I add a new language? Check the document "publi/TuberXpert_add_language.pdf".*

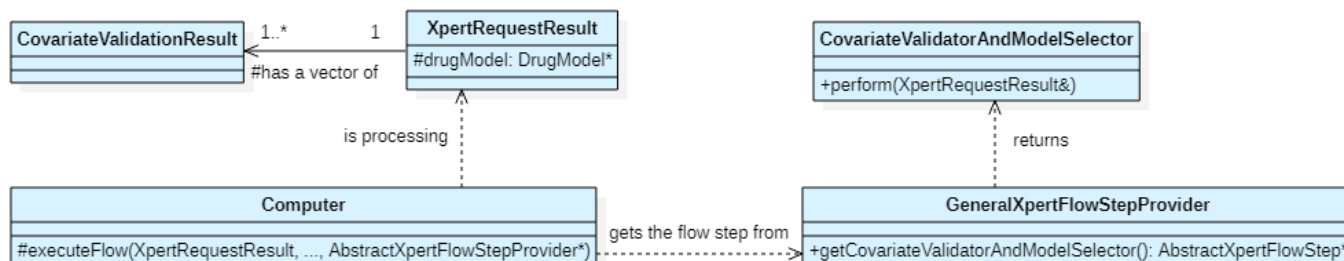## 5.5.2 Validate the covariates and select the best drug model



*Figure 32 - Class diagram of the covariate validation and drug model selection*

The validation of the covariate and the drug model selection involves all these classes. If the execution of the *CovariateValidatorAndModelSelector* class is successful, it places a vector of *CovariatesValidationResult* and the selected drug model in the *XpertRequestResult*.

> *Therefore, we know:*
>
> - *Which drug model best fits the patient's covariates.*
> - *Which covariates the patient does not define. The default value of the definition will be used.*
> - *Which of the patient's covariates are useful for the selected drug model and whether they meet the validation of the corresponding definitions?*

The warning message of a *CovariateValidationResult* is taken directly from the error message of a covariate definition validation in the drug model.

When we generate the final report, to get the covariate information, we will only need to iterate over the vector of *CovariateValidationResult* objects.

> *What about the translation of the name, description and validation error of the covariate? Tucuxi is here to help us. All this information is contained in the covariate definition accessible in each "CovariateValidationResult". When the best drug model is found, we check that each of these values is available in the required language or at least in English. If not, the processing of the "XpertRequestResult" is dropped.*
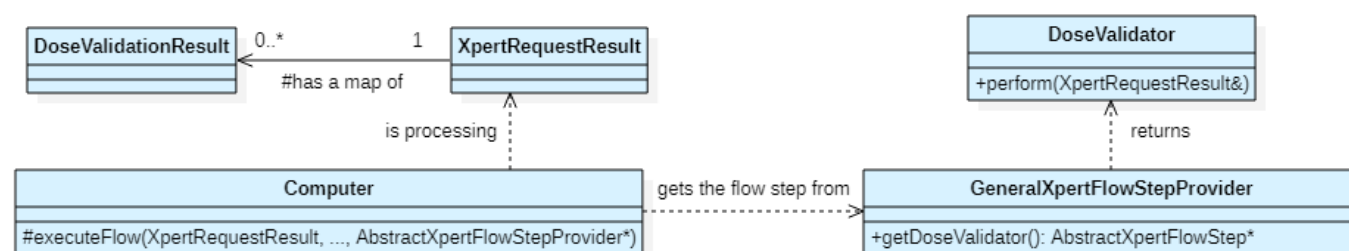
## 5.5.3 Validate the doses

*Figure 33 - Class diagram of the dose validation*

The validation of the doses involves all these classes. If the execution of the *DoseValidator* class is successful, it places a map of *DoseValidationResult* in the *XpertRequestResult*.

> *Why a map and not a vector? Because we do not need a validation result for every dose. We only need a validation for doses that exceed the recommended values. Moreover, to generate the report, it is easier to iterate on the doses of the drug treatment rather than on the validation results. Indeed, in a drug treatment, the doses are not organized in a linear way but in a tree. It is therefore more efficient to prepare and access the validation results.*

The dose warning message is realized by appending the exceeded dose to a sentence from the *LanguageManager*.

For example, if the dose is 500mg: "Maximum recommended dosage reached (400.00 mg)".

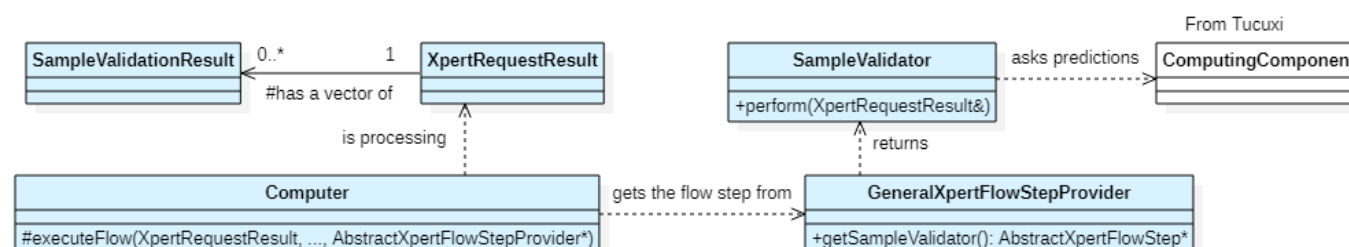> *Therefore, we know which doses need to be verified.*

## 5.5.4 Validate the samples

*Figure 34 - Class diagram of the sample validation*

The validation of the samples involves all these classes. If the execution of the *SampleValidator* class is successful, it places a vector of *SampleValidationResult* in the *XpertRequestResult*.

The only difference with analysis is that one percentile prediction calculation is executed per sample. With only one percentile prediction calculation, if the first and the last sample are really spaced in time, the Tucuxi computation core does not accept to do the calculation because there are too many points to calculate.

> *We therefore know the position of the samples on the 99 percentiles and whether any need to be checked.*

The sample warning message is realized by prepending the percentage of the population that is below or above the suspicious sample percentile to a sentence from the *LanguageManager*.

For example, if the sample percentile is 5: "95% of the population is above this measure".
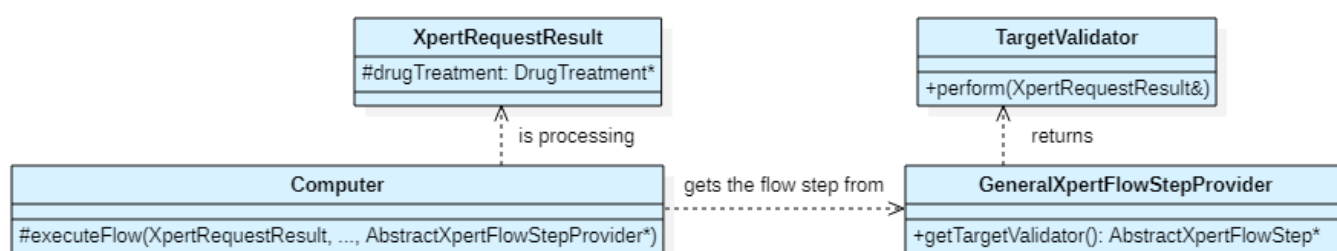
## 5.5.5 Validate the targets



*Figure 35 - Class diagram of the target validation*

The validation of the targets involves all these classes. The *TargetValidator* checks if the targets of the drug treatment in the *XpertRequestResult* are valid. It does not store any results because either the targets are valid and it proceeds to the next flow step or they are not valid and the processing is abandoned.
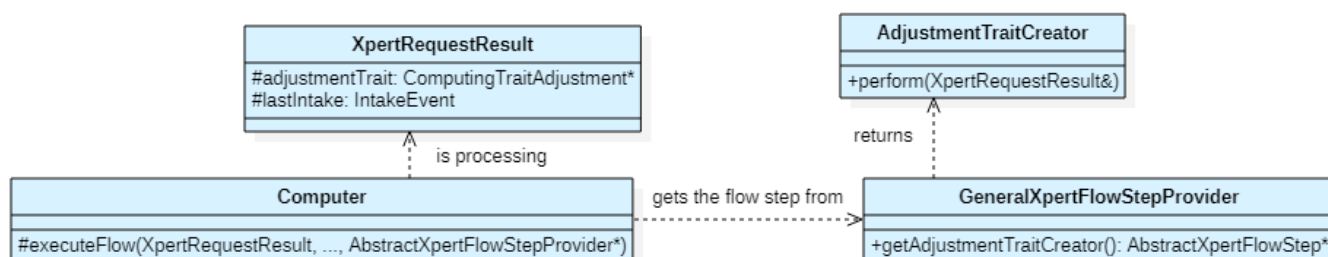
## 5.5.6 Create the adjustment trait



*Figure 36 - Class diagram of the adjustment trait creation*

The creation of the adjustment trait involves all these classes. If the execution of the *AdjustmentTraitCreator* class is successful, it places a *ComputingAdjustmentTrait* in the *XpertRequestResult*. The trait system is what is used to tell the Tucuxi computation core what to predict.

When creating the adjustment trait, the *AdjustmentTraitCreator* must retrieve the list of intakes from the patient's dosing history. We take this opportunity to retrieve the last intake if there is one. This way, we do not need to do the extraction again later.

> *At that point, we have everything we need to calculate the patient's treatment adjustment or first dose.*
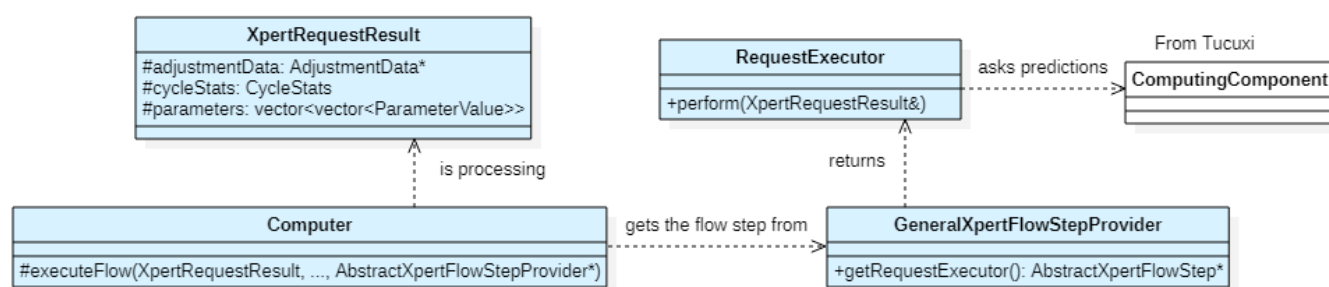
## 5.5.7 Execute the requests



*Figure 37 - Class diagram of the request execution*

The *RequestExecutor* class submits 3 or 4 treatment adjustment predictions to the Tucuxi computation core:

1) The first prediction directly uses the adjustment trait created in the previous step. It aims to obtain the adjustment data which contains for each adjustment found:
   o The predicted concentrations.
   o The times offset.
   o The treatment related to the prediction.
   o The expected values for each target.

   Depending on the type of computation, the pharmacokinetic parameters "A priori" or "A posteriori" of the best adjustment.

2) The second prediction is based on the trait of the first prediction but the end time is made to be on a steady state and the adjustment data are only for the best adjustment. From this resulting adjustment data, we extract the steady state statistics, such as:
   o The peak concentration.
   o The residual concentration.
   o The mean concentration over 24 hours.

3) The third prediction is made only if the first one used "A posteriori" pharmacokinetic parameters. It is also based on the trait of the first prediction but requires "A priori" parameters , the minimum number of points per hour and the adjustment data is only for the best adjustment. From this prediction we get the "A priori" pharmacokinetic parameters.

4) The last prediction trait is similar to the third one but it uses "Typical patient" pharmacokinetic parameters to obtain these parameters.

> *In the current state of the Tucuxi computation core, we are forced to make these additional predictions to get all the information we need. However, these predictions are as light as possible for the computation.*

> *At that point, we have everything we need to generate the report.*
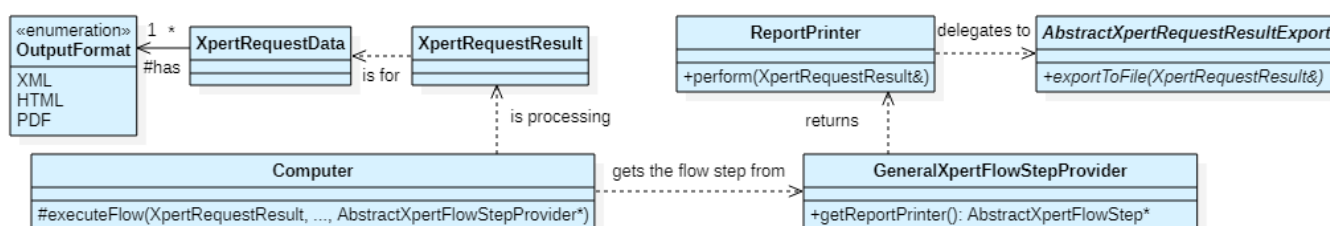
## 5.5.8  Print the report



*Figure 38 - Class diagram of the report generation*

The report generation involves all these classes. The *ReportPrinter* simply creates a concrete instance of an *AbstractXpertRequestResultExport* for the given *OutputFormat* of the *XpertRequestData* in the *XpertRequestResult*. When the corresponding exporter is created, it simply starts the report generation with the *exportToFileMethod*. If the execution of the *ReportPrinter* is successful, the processing of the *XpertRequestResult* is finished. If there are any, the computer can start with a new one.
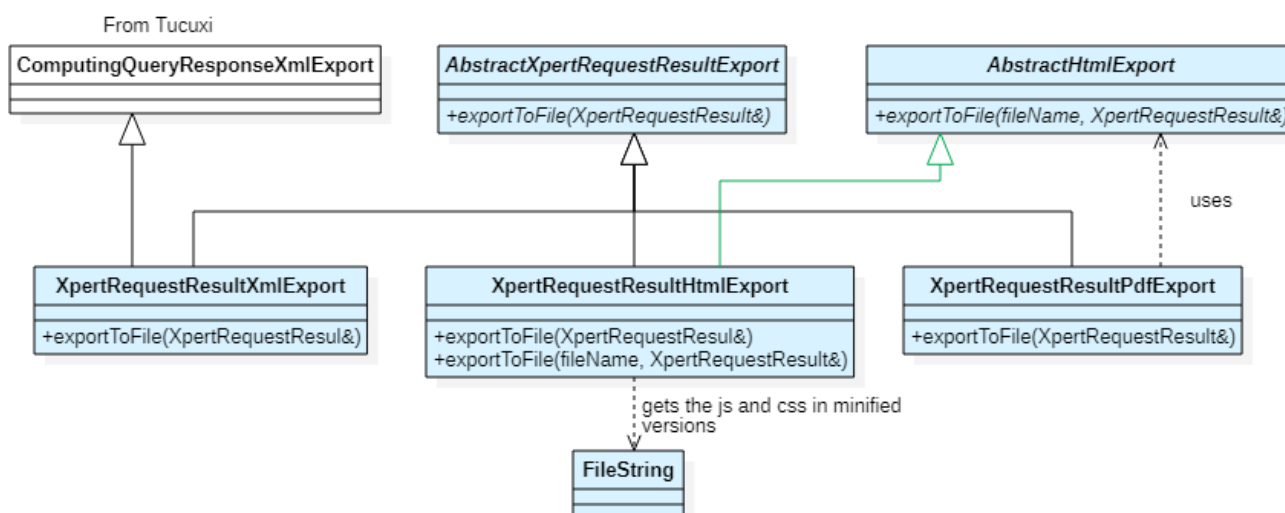
**AbstractRequestResultExport**



*Figure 39 - Class diagram of the request result exporters*

Let's introduce each of these exporters.

**XpertRequestResultXmlExport**

This exporter inherits the Tucuxi exporter because it reuses some of its methods to export the dosage history. Apart from the export of TuberXpert data, this does not add any new logic to what was already there.

The structure of the XML is the same as the one presented [here](here).

> *The structure of the report can be validated with the file "tuberxpert_computing_response.xsd" in "dev/tucuci-tuberxpert/xml/response".*

**XpertRequestResultHtmlExport**

With this export, I did the PDF export at the same time. I have been looking for a way to convert HTML files to PDF files in C++. There are a few solutions but most of them are overloaded, too uncertain for me to risk during this project, not cross-platform or not open source. The best solution I found is to use a project called wkhtmltopdf.

> *The library does dynamic linking. The QMake file in "dev/tucuci-tuberxpert/src/tuberxpert/tuberxpert.pri" automatically switch from Windows « .dll » files to Ubuntu « .a » files.*

It is a C library that uses the Qt WebKit renderer. It is free, open source, cross-platform and always maintained. So, what is the trick? The problem is that even if it is still maintained, it does not support the latest versions of cascading style sheet (CSS) and JavaScript (JS). So, I had to adapt my HTML renderer to be convertible with wkhtmltopdf 0.12.6.

The points of attention are as follows:

- CSS: Many layout options are not supported, such as Flexbox. The HTML layout of TuberXpert has been done only with tables, margins and paddings. However, this is not a problem since we are only trying to make an A4 page (at least the width) in HMTL. So, we can get rid of the responsiveness so that the layout is very simplified.

- JS: JavaScript is only supported in EcmaScript 5 and not in EcmaScript 6. However, it is now possible to write JavaScript and transpile it into older versions.

> *I made a Babel project with Node.js to transpile JS with EcmaScript 6 to EcmaScript 5. The project contains a TuberXpert HTML example and a readme.md that explains how to use it. The project is available in "dev/tucuxi-tuberxpert/html". This is the playground where I designed the report I wanted to have after generating an HTML report with TuberXpert.*

Now that I have a solution to create the PDF report from HTML, I need to find a solution to create an HTML report. To do this, I used a library called Inja 3.3.0 which allows string templating from JSON data objects. It allows loops, conditions, defaults and it is a header only library.

> *Both libraries are located in the folder "dev/tucuxi-tuberxpert/libs". Both libraries support Windows 10/11, Ubuntu 22.04 and they are opensource.*

Basically, the HTML exporter defines a template string, transforms the *XpertRequestResult* data into JSON and that is it. The only thing that differs from the playground project is that the file created is an all-in-one file to be easily manipulated by the user. All the necessary JS and CSS files are minified and placed in static attributes of the *FileString* class.

> *Currently the graphs are drawn by JS scripts provided by Prof. Yann Thoma each time the document is opened. In a future version, it may be more suitable to generate an image, convert it to base 64 and insert it in the final report.*
>
> *The only JS file I created myself is "dev/tucuxi-tuberxpert/html/src/asset/js/tuberxpert.js". This file is responsible for translating the data inserted by the HTML exporter into JS objects used by the graphing library.*

### *XpertRequestResultPdfExport*

Thanks to the previous efforts, the PDF exporter is straightforward. It is inspired by this example of the official wkhtmltopdf GitHub. We are just customizing the margin and disabling the smart shrinking which is messing the final render.

As we expected, the PDF exporter needs to create a temporary HTML file. To do this, we had to make the HTML exporter inherit from *AbstractHtmlExport* to be able to pass a special file name that contains a temporary hint. Normally the exporters create file names automatically from the XpertRequestResult. This new inheritance allows the PDF exporter to give the HTML file a name that indicates that the file is temporary and can be easily retrieved by TuberXpert to delete it.

> *Although, the PDF exporter tries to delete the temporary HTML file, it might be blocked by very aggressive anti-virus software. This way, if deletion is not possible, the user knows that it is not an important file.*
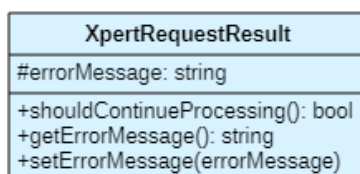
## 5.5.9 Error management



*Figure 40 - Class diagram of the request result with error handling*

This section is not a flow step. It is just there to explain how errors are handled between each flow step.

There are many reasons that can cause a step to fail, but when this happens, the reason detected is logged as an error message in the *XpertRequestResult* and the current process step ends immediately. The TuberXpert computer checks after each flow step if the *XpertRequestResult* can continue with the *shouldContinueProcessing* method. If the method says yes, the *XpertRequestResult* goes to the next step, otherwise the error message is logged and the *XpertRequestResult* is abandoned.

Each event occurring during the execution is recorded in a file called "tuberxpert.log". It lists:

- The import status of the query.
- Which TuberXpert request is currently being processed.
- Which step of the flow is currently running.
- The cause of the failure.

For example, here is the log of a query with only one request that could be fully processed:

```
[22-07-25 18:09:44.412] 33500 info: ****************************************************
[22-07-25 18:09:44.412] 33500 info: Tuberxpert console application is starting up...
[22-07-25 18:09:44.413] 33500 info: --------------------------------------
[22-07-25 18:09:44.413] 33500 info: Processing request number: 1
[22-07-25 18:09:44.413] 33500 info: Checking extraction state...
[22-07-25 18:09:44.413] 33500 info: Extraction succeed.
[22-07-25 18:09:44.413] 33500 info: Loading translation file...
[22-07-25 18:09:44.414] 33500 info: Successfully loaded C:\Users\melvy\Desktop\Dev\2022-herzig\dev\tucuxi-tuberxpert\language/en.xml
[22-07-25 18:09:44.414] 33500 info: Validating covariates and selecting drug model...
[22-07-25 18:09:44.558] 33500 info: Successfully loaded drug model : ch.tucuxi.rifampicin.svensson2017
[22-07-25 18:09:44.558] 33500 info: Covariates validated and drug model selected: ch.tucuxi.rifampicin.svensson2017
[22-07-25 18:09:44.558] 33500 info: Validating doses...
[22-07-25 18:09:44.558] 33500 info: Dosages successfully validated.
[22-07-25 18:09:44.558] 33500 info: Validating samples...
[22-07-25 18:09:51.852] 33500 info: Samples successfully validated.
[22-07-25 18:09:51.852] 33500 info: Validating targets...
[22-07-25 18:09:51.852] 33500 info: Targets successfully validated.
[22-07-25 18:09:51.852] 33500 info: Creating adjustment trait...
[22-07-25 18:09:51.853] 33500 info: Adjustment trait successfully created.
[22-07-25 18:09:51.853] 33500 info: Submission of the adjustment request...
[22-07-25 18:10:06.114] 33500 info: Adjustment request submission succeed.
[22-07-25 18:10:06.114] 33500 info: Generating report...
[22-07-25 18:10:06.132] 33500 info: Report generation succeed.
[22-07-25 18:10:06.132] 33500 info: Tuberxpert console application is exiting...
[22-07-25 18:10:06.132] 33500 info: ****************************************************
```

*Figure 41 - Example of log with success*

For example, here is the log of query with only one request that failed:

```
[22-07-26 12:18:32.725] 20732 info: ****************************************************
[22-07-26 12:18:32.725] 20732 info: Tuberxpert console application is starting up...
[22-07-26 12:18:32.727] 20732 info: --------------------------------------
[22-07-26 12:18:32.727] 20732 info: Processing request number: 1
[22-07-26 12:18:32.727] 20732 info: Checking extraction state...
[22-07-26 12:18:32.727] 20732 info: Extraction succeed.
[22-07-26 12:18:32.727] 20732 info: Loading translation file...
[22-07-26 12:18:32.728] 20732 info: Successfully loaded C:\Users\melvy\Desktop\Dev\2022-herzig\dev\tucuxi-tuberxpert\language/en.xml
[22-07-26 12:18:32.728] 20732 info: Validating covariates and selecting drug model...
[22-07-26 12:18:32.728] 20732 error: The drug files directory does not contain a drug model for the given drug: rifampicin
[22-07-26 12:18:32.728] 20732 info: Tuberxpert console application is exiting...
[22-07-26 12:18:32.728] 20732 info: ****************************************************
```

*Figure 42 - Example of log with failure*
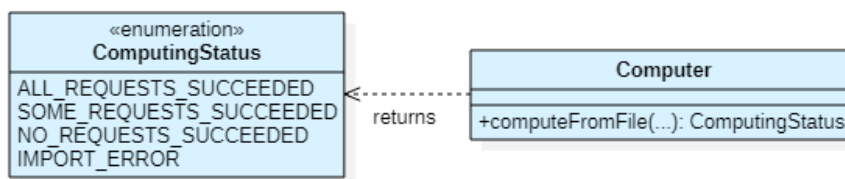
# 5.6 Termination



*Figure 43 - Class diagram of the TuberXpert program ending*

When all the *XpertRequestResult* have been processed, the TuberXpert computer returns a computing status that depends on the number of *XpertRequestResult* that were successfully processed and if the query could be imported. The program exit code depends on the *ComputingStatus* received.

| Exit code | Computing status | Reason |
|:---:|---|---|
| -2 | / | Program arguments are missing. |
| -1 | *IMPORT_ERROR* | The TuberXpert query could not be imported. |
| 0 | *ALL_REQUESTS_SUCCEEDED* | All TuberXpert requests have been fully processed. |
| 1 | *SOME_REQUESTS_SUCCEEDED* | Not all TuberXpert requests have been fully processed. |
| 2 | *NO_REQUESTS_SUCCEEDED* | No TuberXpert request has been fully processed. |

*This concludes this "Implementation" chapter. Again, this chapter is designed to show who does what and when. It is not an in-depth explanation of the code. I think I have made a good effort to document it, so feel free to take a look at the code documentation available in "publi/TuberXpert_code_documentation.html".*

# 6 Tests

This chapter presents the tests that were performed to validate the behavior of the system.

## 6.1 Unit and integration tests

A test program has been set up to verify that TuberXpert works properly during the flow steps, except for the report generation. To do this, it uses the framework Fructose. This framework has been chosen because it is the one already used by Tucuxi.

> *The test program is an independent Qt project available in « dev/tucuxi-tuberxpert/test ».*

The subjects tested are:

- The utility methods.
- The language manager.
- The query importation.
- The query to core extraction.
- The query result creation.
- The covariate validator and the drug model selector.
- The dose validator.
- The sample validator.
- The target validator.
- The adjustment trait creator.
- The request executor.

> *For a complete description of the tests, there is an automatically generated code documentation available in "publi/TuberXpert_test_documentation.html". Each method tells what is being tested and what is expected.*

The organization is simple to understand. There is a test class for each component to test and an additional *TestUtils* class that groups the common elements between the tests:

- Some drug models.
- Date format.
- Unit.
- Translations.
- Functions that set up the environment of execution (2 to 5 of the following points).

In fact, most test cases follow the same pattern:

1. Prepare a query.
2. Import the query.
3. Import some drug models.
4. Import the translation.
5. Create the results holders.
6. Execute the business logic.
7. Compare.

### 6.1.1 Tests results

All the tests are passing

# 6.2 User tests

This section presents the tests that were not automated.

## 6.2.1 Report generation tests

The form of the report generation was tested manually because I found no other practical way to do it programmatically. This section presents what has been verified.

> The directory "dev/tucuxi-tuberxpert/xml/query" contains some queries used to run the upcoming tests. They are prefixed by a number which can be referenced by the column "Query no" in the test description tables.
>
> Sometimes it may be necessary to change the output format or language in the "xpertRequest" element.

**Naming**

| Query no | Description | XML | HTML | PDF |
|---|---|---|---|---|
| Any | The file is named correctly:<br><drug id.>_<xpertRequest no>_<computation time>.<format> | OK | OK | OK |

**Translations**

| Query no | Description | XML | HTML | PDF |
|---|---|---|---|---|
| 2 or 4 | Warnings, covariate names and covariate descriptions are translated into English. (Assuming the selected drug model supports this language) | OK | ╳ | ╳ |
| 2 or 4 | Warnings, covariate names and covariate descriptions are translated into French. (Assuming the selected drug model supports this language) | OK | ╳ | ╳ |
| 2 or 4 | The document is translated into English. (Assuming the selected drug model supports this language) | ╳ | OK | OK |
| 2 or 4 | The document is translated into French. (Assuming the selected drug model supports this language) | ╳ | OK | OK |

**Header**

The expected result is available [here](#).

| Query no | Description | XML | HTML | PDF |
|---|---|---|---|---|
| Any | The computation time of the request is displayed. | OK | OK | OK |
| Any | The language of the "xpertRequest" element is displayed. | OK | ╳ | ╳ |
| Any | The drug identifier is displayed. | OK | OK | OK |
| 2 or 4 | The last dose value and unit are displayed. | OK | OK | OK |
| 1 or 3 | The last dose element is empty. | OK | ╳ | ╳ |
| 1 or 3 | The last dose is represented by a "-". | ╳ | OK | OK |
| Any | The drug model identifier is displayed. | OK | OK | OK |

## Admin

The expected result is available here.

| Query no | Description | XML | HTML | PDF |
|---|---|---|---|---|
| 1 | The admin element is not present. | OK | | |
| 1 | The contact table values are "-" and the clinical data display is "none". | | OK | OK |
| 2 | (Complete) The admin element is as entered. | OK | | |
| 2 | (Complete) There are no "-" values and the clinical data are displayed. | | OK | OK |
| 3 | (Minimal person and institute) The admin element is as entered. | OK | | |
| 3 | (Minimal person and institute) The current values are correctly displayed. Missing values are "-" and the display of clinical data is "none". | | OK | OK |
| 4 | (Minimal address, phone and email) The admin element is as entered. | OK | | |
| 4 | (Minimal address, phone and email) The present values are correctly displayed. The identifiers are "-" and the clinical data display is "none". Addresses (state/country), phones (type) and emails (type) are not shown. | | OK | OK |

## Covariates

The expected result is available here.

| Query no | Description | XML | HTML | PDF |
|---|---|---|---|---|
| 2 or 4 | The patient covariates and the definitions of the covariates that replace the missing patient covariates are all displayed. They are sorted by name. The names are sorted according to the C++ comparison. This means that 'Z' < 'Â'. | OK | OK | OK |
| 2 | Boolean values and gender are text values instead of doubles. | | OK | OK |
| 2 | The covariates are complete. The warning is only present when needed and the date when the covariate is from the patient | OK | OK | OK |

## Treatment

The expected result is available here.

| Query no | Description | XML | HTML | PDF |
|---|---|---|---|---|
| 1 to 4 | The structure of the dose history is as entered, but the dosage time ranges are sorted in ascending order of start time. | OK | | |
| 1 or 3 | The treatment display is "none". | | OK | OK |
| 2 or 4 | The time ranges are correctly displayed with all their doses that have an administration route and a posology. The time ranges are sorted in ascending order of start time. | | OK | OK |
| 2 or 4 | A warning is associated to a dose when needed. | OK | OK | OK |

**Samples**

The expected result is available here.

| Query no | Description | XML | HTML | PDF |
|---|---|---|---|---|
| 2 or 4 | The samples are all displayed. They are sorted by date of measurement. | OK | OK | OK |
| 2 | The samples are complete. The warning is only present when needed. | OK | OK | OK |
| 1 or 3 | The samples element is empty. | OK | ✕ | ✕ |
| 1 or 3 | The samples display is "none. | ✕ | OK | OK |

**Best adjustments and suggested adjustment**

The expected result is available here.

| Query no | Description | XML | HTML | PDF |
|---|---|---|---|---|
| Any | There is at least one adjustment element in the dataAdjustment element. They are sorted by score. The elements respect the specified structure. | OK | ✕ | ✕ |
| 1 or 2 | The "adjustments found" section is displayed correctly: Introductory sentence, graph (with all the adjustments predictions and the targets) and the adjustment dosage history (follows the same structure as the treatment part but there is the adjustment score in addition to each dosage history.) | ✕ | OK | OK |
| 3 or 4 | The "adjustments found" section is not displayed because there is only one adjustment found. | ✕ | OK | OK |
| 3 or 4 | The "adjustment suggested" section is displayed correctly: Introductory sentence, graph (with the best adjustment predictions and the targets) and the best adjustment dosage history (follows the same structure as the treatment part but there is the adjustment score in addition to the dosage history.) | ✕ | OK | OK |
| Any | The targets are correctly displayed. | ✕ | OK | OK |

**Computation facts**

The expected result is available here.

| Query no | Description | XML | HTML | PDF |
|---|---|---|---|---|
| Any | The parameters element contains the "Typical patient" parameters. | OK | ✕ | ✕ |
| Any | The pharmacokinetic parameters section displays the parameters of the typical patient. | ✕ | OK | OK |
| 1 or 3 | The parameters element contains the parameters "A priori" but not "A posteriori". | Ok | ✕ | ✕ |
| 1 or 3 | The pharmacokinetic parameters section displays "A priori" but not "A posteriori" parameters. | ✕ | OK | OK |

| | | | | |
|---|---|---|---|---|
| 2 or 4 | The parameters element contains the parameters "A priori" and "A posteriori". | OK | | |
| 2 or 4 | The pharmacokinetic parameters section displays "A priori" and "A posteriori" parameters. | | OK | OK |
| Any | The steady state statistics are correctly displayed. | OK | OK | OK |
| Any | The elements of the computation covariates are correctly realized. | OK | | |
| Any | The calculation covariates are correctly displayed. Their name is used instead of the identifier. | | OK | OK |
| 2 | Boolean values and gender are text values instead of doubles. | | OK | OK |

## 6.2.2 Error handling tests

This series of tests attests to the good behavior of TuberXpert in case of processing failure.

*The directory "dev/tucuxi-tuberxpert/xml/query/failure_tests" contains some queries used to run the upcoming tests. They are prefixed by a number which can be referenced by the column "Query no" in the test description tables.*

*When it says that something "is logged", it refers to the fact that it is logged in the "tuberxpert.log" file which lists every event during the runs.*

| Query no | Description | Result |
|---|---|---|
| / | When a program argument is missing, it displays the required arguments in the console. | OK |
| 5 | When the query cannot be imported, the problem is logged and the return code is -1. | OK |
| 6 | When the query cannot be extracted, the problem is logged and the return code is 2. | OK |
| Any | (By removing the translations files path as argument) When translations cannot be loaded, the problem is logged and the return code is 2. | OK |

*From this point, we assume that the "ch.tucuxi.imatinib.gotta2012.tdd" model is the only drug model for imatinib.*

| | | |
|---|---|---|
| 7 | When the covariate validator and model selector fail, the problem is logged and the return code is 2. | OK |
| 8 | When the dose validator fails, the problem is logged and the return code is 2. | OK |
| 9 | When the sample validator fails, the problem is logged and the return code is 2. | OK |
| 10 | When the target validator fails, the problem is logged and the return code is 2. | OK |
| 11 | When the adjustment trait creator fails, the problem is logged and the return code is 2. | OK |

> *The following test depends mainly on the system on which it is run. Normally, no request can make the report generator fail, because if we start the report generation, the request is supposed to be correct. However, sometimes files cannot be created or deleted. For example, on my computer, the pdf report cannot be created if there is already a file with its name. This is due to my anti-virus.*

| None | When the adjustment trait creator fails, the problem is logged and the return code is 2. | OK |
|------|----------------------------------------------------------------------------------------|-----|

### 6.2.3  Tests results

All the tests are passing

# 6.3 Known issues

- Tucuxi does not really support dates that are too old. For example, the imatinib drug model requires the patient's age to be between 20 and 88. If the calculation time is 2022 and the patient's date of birth is 1900, it works fine, the constraint is not met. However, if the date of birth is 1400, the age appears to meet the constraint.

- If a TuberXpert request indicates that the target extraction option is "individualTargets" and the query does not specify any targets, the adjustment data contains some adjustments but they have no target. TuberXpert expects the adjustments to always have at least one target when the report is generated in HTML or PDF. In this case, the program interrupts the processing of the corresponding TuberXpert request.

# 7 <u>Conclusion</u>

To conclude this thesis, I will talk about TuberXpert through three axes: the planning, the software and my personal feelings.

## 7.1 Planning perspective

*The plannings are available [here](here).*

In general, I would say that the planning was not so bad.

Of course, the initial planning was a bit naive in the sense that I forgot some steps, some were overestimated and some were underestimated. I think the main problem I had at this point was that my idea of TuberXpert was not representative of what I was actually going to do. In my mind, there would have been more "pharmacology" decisions to make, like "if the patient has this tendency, we can make a prediction to test this hypothesis. If the hypothesis is tested, we can try this approach by making this prediction". I was dreaming. Realistically, it was outside my expertise and would have taken much more time than was allocated to this thesis. Also, it was a very bad idea and bad practice to schedule the tests at the end. If something was really not working, the deadline would have been too close to make major changes.

As for the intermediate planning, since I had already immersed myself in the topic, the horizon of necessary tasks was clearer because all the analysis was already done. The only downside to this planning is that I overestimated the time needed to generate the reports and was naive to think that I would have done the documentation at the same time I was doing the implementation. In fact, it saved me a lot of time, because it was not uncommon for me to go back and change things because they were not really well implemented, there was a kind of iteration in the way I was programming, like in agile programming. So, it saved me a lot of time not having to go back and forth adjusting the documentation to reflect the actual state of the program. However, next time, I should still find a better balance between creating the documentation along the way and completing it at the end of the project.

In my opinion, what made this planning successful was the fact that everything went smoothly. I did not run into any unforeseen circumstances that made me rethink everything I had planned to do. Do not get me wrong, I do not think it was a matter of luck, but rather because everything I did was carefully analyzed and planned. At no point I dove into something unknown without knowing where I was really going.

To summarize, I think successful planning is always having an eye on the future, anticipating what comes next, and not being blinded by the present. In that sense, I think the way I managed TuberXpert was quite good.

## 7.2 Software perspective

If we think back to the software specification, the job is done. All functionalities could be successfully implemented. TuberXpert can validate patient data if it is suspicious, it provides a first treatment or some adjustments, it generates clear reports in multiple formats with graphs, it supports several languages and it is cross-platform. Put like that, it does not sound like much but it is, moreover, if we think about what comes with the software: testing, code documentation, project documentation. I think we have a solid first version of TuberXpert, which is really encouraging.

Speaking of evolution, I really tried to make TuberXpert flexible. I am convinced that it can be easily enriched with new features and adapted to any drug. I mean, it is already designed for all drugs, but it can become more tailored to some of them if necessary. With the factory systems and flow steps, TuberXpert can really be modulated.

Some say we cannot assume our program works until we test it. With that in mind, I am not taking much of a risk in saying that TuberXpert "seems" to work. Everything except the report generation, because I have not had time to do that, is tested automatically. On the other hand, the report generation is tested by the user. I agree that this is only acceptable for a temporary solution. In the future, the tests for the reports should be automated.

Although TuberXpert is already an amazing project in my eyes, I think the best is yet to come. I am looking forward to seeing its evolution.

## 7.3 Personal feelings perspective

At the end of this thesis, I feel really enriched. I had the chance to work on a theme that made me learn something new. In order to develop TuberXpert, I had to study the basics of therapeutic drug monitoring and pharmacology. As someone who likes to discover new things, this was a great opportunity because I do not think I would have known this field in any other context.

It was very exciting to know that I was working on something useful that would be further developed into something bigger. For the first time in my software engineering studies, I felt like I was working for something important, when no such motivation exists. Some Bachelor's theses will gather dust in a repository but not TuberXpert and that is a real satisfaction.

In addition, TuberXpert made me work on a new type of project because I never had to work with an existing code base. To implement TuberXpert, I had to explore and understand the implementation of Tucuxi. In my opinion, this is common practice for a developer, but it is not something we train at school. So, it was great for me to do this experience for the first time.

To conclude this thesis, I would say that I am very honored to be one of the developers of TuberXpert and I am proud of the work I am presenting.

# 8  <u>Thanks</u>

I would like to thank especially Professor Yann Thoma for having supervised me throughout this Bachelor's thesis and I also thank the expert Jonas Chapuis.

# 9 Bibliography

- BENALLAL Nadir, 2018, Expert System for drug dosage adaptation [PDF document], Yverdon-les-Bains: Haute école d'ingénierie et de gestion du canton de Vaud. Bachelor's thesis

- BUCLIN Thierry, 2022, Les Bases de la pharmacocinétique clinique [PDF document]

- CADUFF Max, 2020, Expert System for drug dosage adjustments [PDF document], Yverdon-les-Bains: Haute école d'ingénierie et de gestion du canton de Vaud. Bachelor's thesis

- Prof. THOMA Yann, prof. MPAGAMA Stellah, GUIDI Monia, 2022, TuberXpert: Clinical Decision Support System for antituberculosis medical drugs [PDF document]

- TUCUXI dev team, 2022, Tucuxi CLI Software Usability Specification [PDF document]

- TUCUXI dev team, 2022, Tucuxi Drug Model File Specification [PDF document]

- TUCUXI dev team, 2022, Tucuxi System Description [PDF document]

- Datapharm, 2022, Rifampicin 150 mg Capsules. In EMC web site [online], updated 21 Feb. 2022 [24 Mar. 2022], https://www.medicines.org.uk/emc/product/8788/smpc

- Report icons, 2022, Freepik collection. In flaticon web site [online], [23 March 2022], https://www.flaticon.com/fr/auteurs/freepik

- Report icons, 2022, Gajah Mada collection. In flaticon web site [online], [23 March 2022], https://www.flaticon.com/fr/auteurs/gajah-mada

- World Health Organization, 2022, Tuberculosis fact sheet. In: World health Organization web site [online], updated 14 Oct 2021 [25 July 2022], https://www.who.int/news-room/fact-sheets/detail/tuberculosis

# 10  <u>Authentication</u>

The undersigned, Melvyn Herzig, hereby certifies that he alone conducted this work and did not use any other source than those expressly mentioned.

Yverdon, the Sunday, May 15, 2022

Melvyn Herzig

# 11  <u>List of abbreviations</u>

**ATC** *anatomical therapeutic chemical*

**CDSS** *clinical decision support system,*
**CLI** *command-line interface*
**CSS** *cascading style sheet*

**DM** *diabetes mellitus*

**GUI** *graphical user interface*

**HIV** *human immunodeficiency viruses*
**HTML** *hypertext markup language*

**IDE** *integrated development environment*

**JS** *JavaScript*

**PDF** *portable document format*

**TB** *tuberculosis*
**TDM** *therapeutic drug monitoring*

**XML** *extensible markup language*

# 12   <u>List of figures</u>

# 13  Annexes

As explained at the very beginning, this documentation was written assuming that the reader has access to the project repository. Therefore, the real annex of this report is the compressed archive of the repository.

The main points of interest are:

- In "tiersdoc":
    - Tucuxi_CLI_Usability_Specification.pdf

- In "publi":
    - TuberXpert_add_language.pdf
    - TuberXpert_code_documentation.html
    - TuberXpert_test_documentation.html
    - tb_Herzig_Melvyn_2022.pdf (this document)
    - "/example_reports" for examples of reports

- In "dev/tucuxi-tuberxpert":
    - "/html"  prototype of HTML report and JavaScript transpiler
    - "/src" C++ source code of TuberXpert
    - "/test" TuberXpert automated test project
    - "/xml" TuberXpert query and validation files for the query and the report

# 14  Planning

This chapter showcases the Gantt charts of the planning. The initial diagram was made after a couple of hours of work, the intermediate diagram on 16.05.2022 (≈186h) and the final diagram on 29.07.2022 (≈526 h).

To get an idea of the workload, the work was divided into two main parts:

- From 24.02.2022 to 17.06.2022
  In the final semester of classes, 16 weeks part-time, 248h ≈ 15.5h per week.

- From 18.06.2022 to 29.07.2022
  During summer vacation, 6 weeks full time, 278h ≈ 46.3h per week.
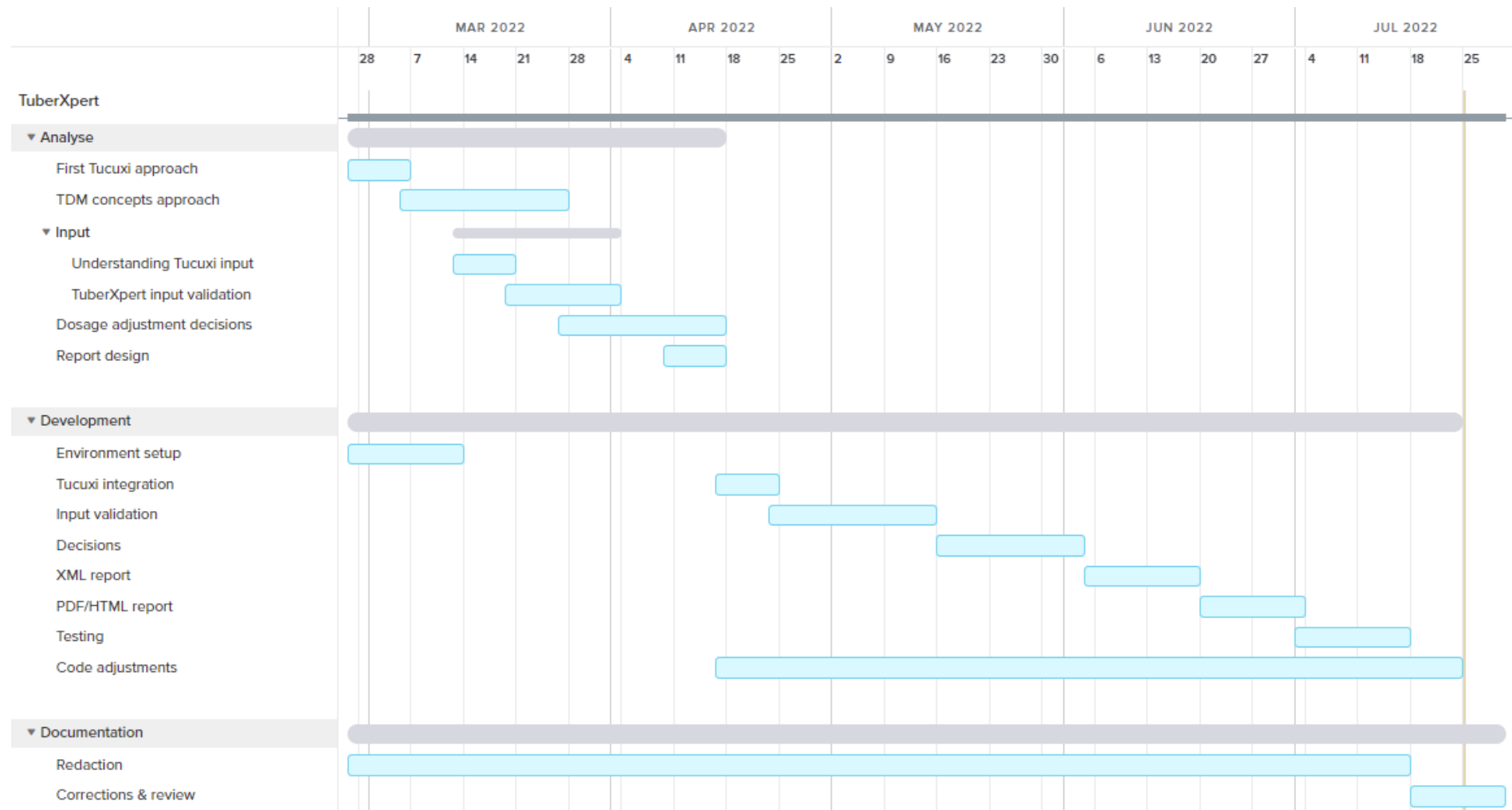
## 14.1 Initial



*Figure 44 - Gantt chart of the initial planning*
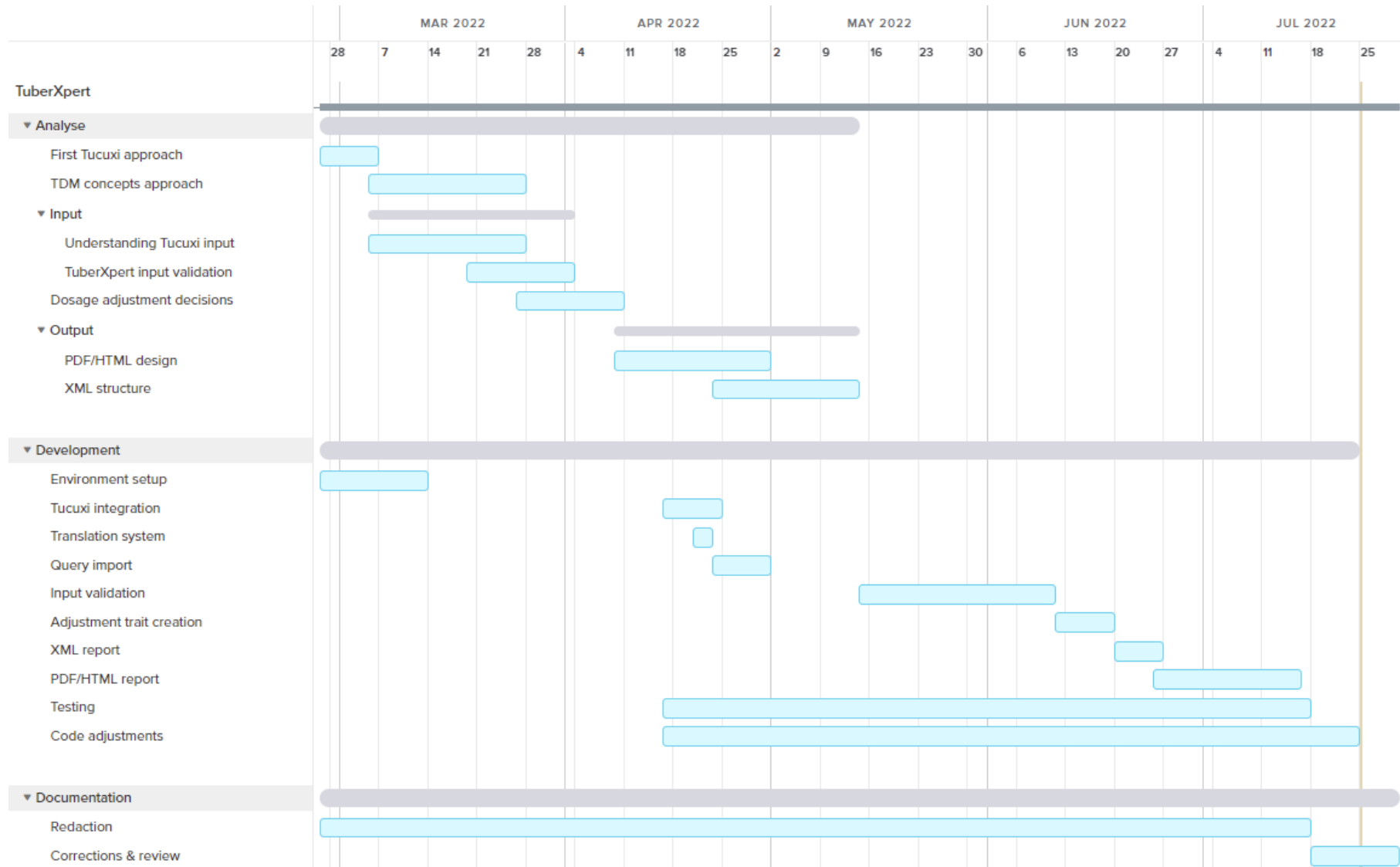
## 14.2 Intermediate
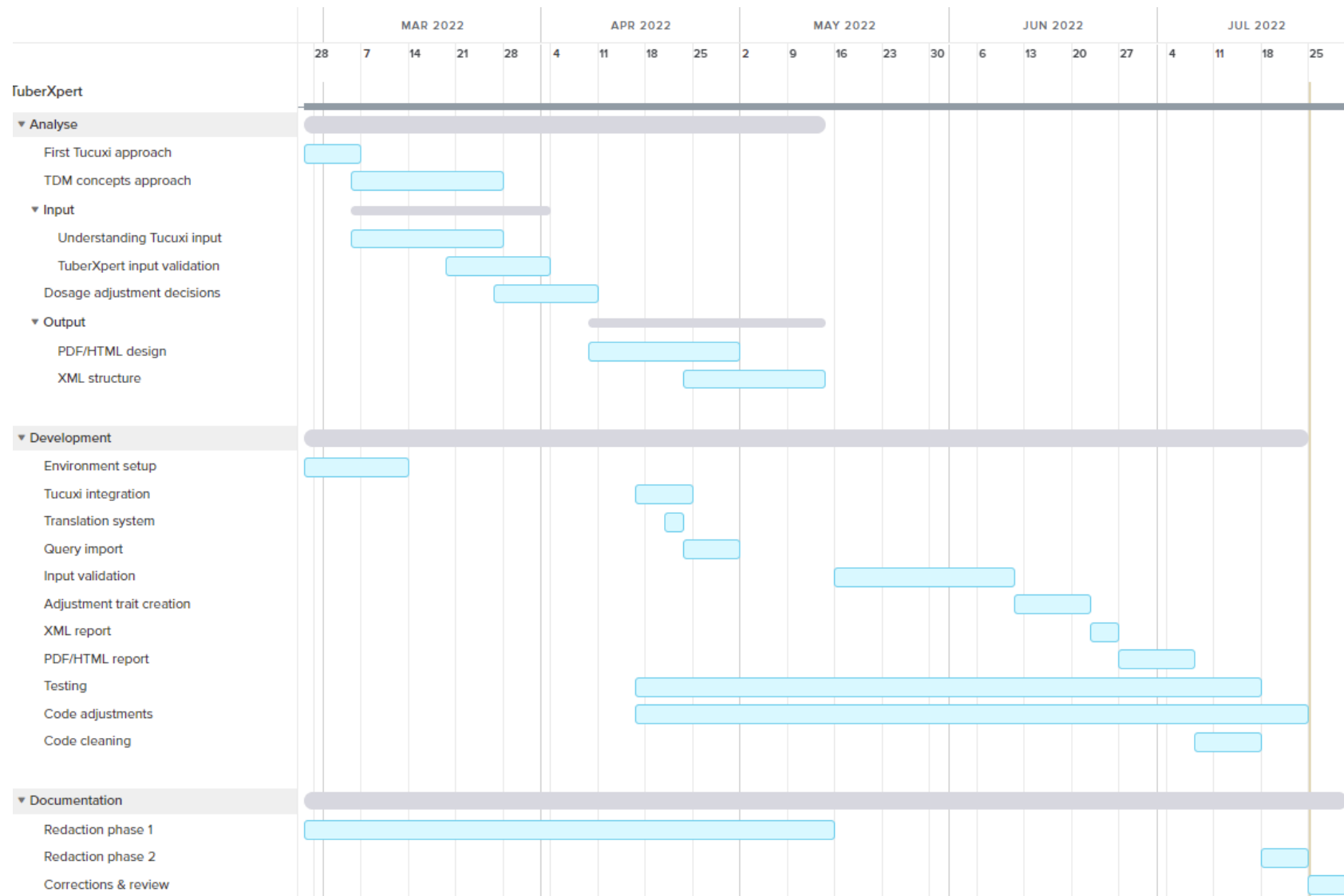


*Figure 45 - Gantt chart of the intermediate planning*

## 14.3 Final



*Figure 46 - Gantt chart of the final planning*